UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DESARROLLO DE LA TELEOPERACIÓN DE ROBOTS INDUSTRIALES Y COLABORATIVOS MEDIANTE TÉCNICAS AVANZADAS DE VISIÓN ARTIFICIAL

Trabajo Fin de Máster en Automática e Informática Industrial

Autora: Emima Ioana Jiva

Tutor: Ángel Valera Fernández

Cotutora: Marina Vallés Miquel

Departamento de Ingeniería de Sistemas y Automática
Universitat Politècnica de València

Valencia, Julio 2019

RESUMEN

Actualmente la robótica está teniendo un crecimiento muy grande, siendo la piedra angular de lo que se conoce como la Industria 4.0. Para facilitar la implantación de los sistemas robotizados en la industria general, y la pequeña y mediana empresa en particular, los robots deben tener una serie de características: deben ser económicos, fáciles de utilizar, versátiles, etc.

En este Trabajo Fin de Máster (TFM) se propone desarrollar una aplicación para que pueda teleoperar sistemas robotizados. Para que sea lo más eficiente posible, se debe permitir que la interacción entre el operario y el robot sea lo más directo y natural posible. Además, debería permitir la teleoperación de diferentes sistemas robotizados.

Para permitir una interacción natural, en este TFM se propone utilizar técnicas avanzadas de visión artificial. Gracias a estas técnicas, la cámara de visión detectará la posición y los movimientos del brazo del operario, de forma que se mandará dicha información a la unidad de control del robot para generar las referencias de movimiento de los robots.

Los robots a utilizar en este TFM son los robots disponibles en el Dpto. de Ingeniería de Sistemas y Automática y el Instituto U. de Automática e Informática Industrial. En concreto se trabajará con del robot IRB140, un robot industrial compacto de 6 grados de libertad. Además, se deberá programar y controlar también el robot YUMI, un robot colaborativo de última generación compuesto por 2 brazos y un total de 14 grados de libertad.

Palabras clave: robótica industrial, robótica colaborativa, control de robots, visión artificial, interfaces hombre-máquina

RESUM

Actualment la robòtica està tenint un creixement molt gran, sent la pedra angular del que es coneix com la Indústria 4.0. Per a facilitar la implantació dels sistemes robotitzats en la indústria general, i la xicoteta i mitjana empresa en particular, els robots han de tindre una sèrie de característiques: han de ser econòmics, fàcils d'utilitzar, versàtils, etc.

En aquest Treball Fi de Màster (TFM) es proposa fer una aplicació perquè puga teleoperar sistemes robotitzats. Perquè siga el més eficient possible, s'ha de permetre que la interacció entre l'operari i el robot siga el més directe i natural possible. A més, hauria de permetre la teleoperació de diferents sistemes robotitzats.

Per a permetre una interacció natural, en aquest TFM es proposa utilitzar tècniques avançades de visió artificial. Gràcies a aquestes tècniques, la càmara de visió detectarà la posició i els moviments del braç de l'operari, de manera que es manarà la dita informació a la unitat de control del robot per a generar les referències de moviment dels robots.

Els robots a utilitzar en aquest TFM són els robots disponibles en el Dpto. d'Enginyeria de Sistemes i Automàtica i l'Institut U. d'Automàtica i Informàtica Industrial. En concret es treballarà amb del robot IRB140, un robot industrial compacte de 6 graus de llibertat. A més, s'haurà de programar i controlar també el robot YUMI, un robot col·laboratiu d'última generació format per 2 braços i un total de 14 graus de llibertat.

Paraules clau: robòtica industrial, robòtica col·laborativa, control de robots, visió artificial, interfícies home-màquina

ABSTRACT

Currently, robotics are having a very large growth, being the cornerstone of what is known as Industry 4.0. To facilitate the implementation of robotic systems in general industry, and small and medium enterprises in particular, robots must have a number of characteristics: they must be economical, easy to use, versatile, etc.

In this Final Master's Project (TFM) it is proposed to develop an application so that it can teleoperate robotic systems. To be as efficient as possible, the interaction between the operator and the robot must be allowed to be as direct and natural as possible. In addition, it should allow teleoperation of different robotic systems.

To allow a natural interaction, in this TFM it is proposed to use advanced artificial vision techniques. Thanks to these techniques, the vision camera will detect the position and movements of the operator's arm, so that this information will be sent to the robot control unit to generate the movement references of the robots.

The robots to be used in this TFM are the robots available in the Dept. of Systems Engineering and Automation and the U. Institute of Automation and Industrial Computing. In particular we will work with the IRB140 robot, a compact industrial robot with 6 degrees of freedom. In addition, the YUMI robot, a last-generation collaborative robot composed of 2 arms and a total of 14 degrees of freedom, must also be programmed and controlled.

Keywords: industrial robotics, collaborative robotics, robot control, artificial vision, man-machine interfaces

ÍNDICE DEL CONTENIDO

ÍNDICE DE LAS IMÁGENES

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Trabajo Fin de Máster en Automática e Informática Industrial

DOCUMENTO 1: MEMORIA

Departamento de Ingeniería de Sistemas y Automática

Universitat Politècnica de València

Valencia, Julio 2019

1. INTRODUCCIÓN

1.1. INTRODUCCIÓN Y MOTIVACIÓN

En los últimos años, la tecnología ha ido adquiriendo importancia en todos los sectores de la vida. Entre los avances más importantes que se han hecho a lo largo de los años, es el avance en el ámbito de la robótica. Hoy en día, la robótica está presente principalmente en la vida cotidiana de las personas, así como también, en el mundo industrial. Esta rama tecnológica puede llegar a sustituir el trabajo difícil del ser humano ahorrando de esta forma tiempo y dinero en la realización de las tareas.

Actualmente, se puede observar que hay muchas aplicaciones de sistemas robotizados, como, por ejemplo, drones, robots domésticos, coches autónomos, etc. Con todo esto, se puede decir que la robótica está sujeta a un constante cambio y evolución, pudiéndose desarrollar infinidad de aplicaciones con el fin de hacer que las tareas realizadas por las personas sean más fáciles.

Otro de los sectores que está en pleno desarrollo son las comunicaciones inalámbricas, ya que el hecho de no utilizar cables para realizar conexiones hace que sea posible la transmisión de datos de un dispositivo a otro a una distancia bastante elevada.

En este proyecto se intenta aprovechar estos últimos avances de la tecnología y se ha deseado diseñar una aplicación en la cual un robot pueda ser tele operado a través de la imitación del movimiento de los brazos de una persona.

1.2. OBJETIVO DEL TRABAJO

Existen ocasiones en las que una persona realiza trabajos peligrosos para la vida humana, como es el caso de ciertos departamentos de las centrales nucleares, pero que sin la presencia de esa persona la actividad no se puede llevar a cabo ya que se necesita que alguien realice ciertos movimientos. En otras ocasiones, las personas se encuentran fuera de su lugar de trabajo y existen emergencias en las que se tienen que desplazar para solucionar el problema.

Estos son solo algunas de los problemas que se pretenden solucionar con el proyecto que se explica a continuación.

El objetivo final de este proyecto es realizar una aplicación en la cual un robot copie los movimientos que una persona realiza con sus brazos. De esta manera, en el caso de una persona que trabaja en zona peligrosa, la persona mueve sus brazos sin estar en esa zona y el robot copia sus movimientos estando en la zona peligrosa. Así pues, se conseguirá realizar el trabajo sin que una persona se encuentre en peligro.

Para detectar los movimientos de los brazos de la persona que tele opera se desarrollará una aplicación basada en visión artificial que obtendrá información de estos movimientos y la enviará al robot.

Con el fin de alcanzar este objetivo final se han planteado otros objetivos parciales:

- Investigar sobre los marcadores Aruco con el fin de poder entender cómo funcionan y el tamaño que debe tener para que estando a una distancia de la cámara el programa sea capaz de detectarlos.
- Adquirir conocimientos sobre el lenguaje de programación C++ y sobre OpenCV
- Realizar la comunicación entre el programa que detecta los marcadores y el robot.
- Conseguir que el robot mueva sus dos brazos a la vez utilizando la información proporcionada por el programa que detecta los marcadores.

2. PARTE TEÓRICA

2.1. ROBÓTICA

Hace ya miles de años, los seres humanos empezaron a construir máquinas que imitaban parte del cuerpo humanos con el fin de que estas puedan hacer los trabajos más pesados que los debía hacer un ser humanos. Así pues, en el año 1923 Karel Capek inventó el término de "robot" a partir de la palabra *robota,* que significa trabajo forzado.

Hoy en día, según la RAE, podemos definir el concepto robot como: "Máquina o ingenio electrónico programable capaz de manipular objetos y realizar operaciones antes reservadas solo a personas" [16]. Años más tarde, aparece el término de robótica que se podría definir como la rama tecnológica que se encarga del diseño, construcción y operación de los robots [17].

En la actualidad existen diversas clasificaciones para robots, todo depende del criterio que se utilice. En este trabajo se ha considerado adecuado clasificar los robots según su funcionalidad. Según esta clasificación existen dos grandes grupos: los robots manipuladores y los robots móviles.

2.1.1. Robots manipuladores

La principal característica de estos robots es que son sedentarios. Esto significa que son robots estructurados para moverse en un determinado espacio de trabajo, según uno o más sistemas de coordenadas y con un número limitado de grados de libertad. También suelen recibir el nombre de robots poli articulados.

En general, tienen una configuración de brazo articulado y se utilizan con mucha frecuencia en la industria para procesos de carga y descarga de materiales, soldadura o manipulación de objeto

Este tipo de robots utilizan diferentes tecnologías para funcionar y por tanto presentan diversos niveles de complejidad [18]:

- En el nivel más bajo se encuentran los robots que se mueven con la ayuda de la intervención humana.

- En el nivel un poco más avanzado se encuentran los robots que no necesitan la intervención humana para trabajar. Estos se guían por posiciones fijas. Este tipo de robots necesitan que la ubicación y el entorno de trabajo sea siempre el mismo ya que si se cambia ya no serán capaces de realizar las actividades para las cuales estaban diseñados. Generalmente utilizan una estructura de control abierto, esto quiere decir que el robot no recibe información sobre cómo se está realizando el trabajo por lo tanto el robot no modificará su comportamiento.

- En el nivel más alto se encuentran los robots que utilizan una estructura de control cerrada. En este tipo de robots, existe una retroalimentación del exterior y de esta forma pueden modificar su funcionamiento en función de los valores que recibe. Normalmente se utilizan los sensores para proporcionar la información del entorno.

En esta categoría, se suelen incluir también los robots industriales ya que presentan las mismas funcionalidades. Estos robots tienen varias definiciones:

La Asociación de Industrias de Robótica (RIA) define al robot industrial como: "manipulador multifuncional reprogramable, capaz de mover materias, piezas, herramientas o dispositivos especiales, según trayectorias variables, programadas para realizar diversas tareas" [19].

La Organización Internacional de Estándares (ISO) por su parte, define al robot industrial como:" manipulador programable con varios grados de libertas, capaz de manipular materias, piezas, herramientas o dispositivos especiales según trayectorias variables programadas para realizar diversas tareas" [20].

En una de las partes de este proyecto se ha utilizado el robot manipulador IRB 140 de la empresa ABB.

Fig.2.1 Robot manipulador IRB 140 [1]

2.1.2. Robots móviles

La característica que les diferencian de la categoría anterior es la capacidad que tienen para desplazarse. Por tanto, utilizan mecanismos para la transmisión de movimiento. Estos robots tienen un mayor grado de libertad y por tanto puede realizar mejor algunas funciones que son propias de los seres vivos.

Para controlar su trayectoria y su velocidad existen muchas formas. Hay robots que se controlan con mandos e incluso algunos sensorizados que son capaces de realizar su tarea sin necesidad de intervención humana.

De igual forma que los robots manipuladores, los robots móviles se pueden clasificar de diversas formas. A continuación, se explicará una clasificación en función del medio en el que los robots son capaces de desplazarse [13].

- Aire: Tienen capacidad de desplazarse por el aire de igual forma que un avión o un helicóptero lo puede hacer. La configuración que más se utiliza es la de cuatro hélices. En la actualidad existen importantes aplicaciones desarrolladas con este tipo de robots, como es el caso de los drones utilizados para el transporte de mercancías o los que sirven platos a los clientes en los restaurantes.

Fig.2.2 Robot móvil de aire [2]

- Agua: Tienen capacidad de desplazamiento en medios acuáticos. Este tipo de robots pueden entrar en lugares donde el ser humano no puede acceder con facilidad, como por ejemplo las altas profundidades. Así pues, el desarrollo de aplicaciones con este tipo de robots también es interesante ya que se podría conseguir realizar actividades que al ser humano le es difícil o incluso imposible.

- Tierra: Son los más abundantes y a su vez se pueden clasificar en:

    - Robots con ruedas [13]:

  Los más utilizados son los que tienen configuración diferencial: dos ruedas opuestas en un mismo eje que debe ser perpendicular a la dirección de avance del robot. En el otro extremo se sitúa una rueda que gira libremente sin estar relacionada con ningún motor. Para generar una trayectoria se debe variar las velocidades de los motores.

  La configuración triciclo es igual que la anterior con la diferencia de que en este caso la tercera rueda es orientable y por tanto para variar la trayectoria se actúa sobre la orientación de esta rueda.

  Otra configuración es la llamada Ackerman constituida por dos ruedas traseras con tracción en el mismo eje perpendiculares a la dirección de avance y dos ruedas delanteras para la dirección.

Fig.2.3 Robot móvil con ruedas [3]

- Robots andadores [13]

Este tipo de robots imitan a humanos y animales. Presentan patas que las utilizan para mantenerse en posiciones de equilibrio. Los más conocidos son los robots bípedos o los hexápodos.

Fig.2.4 Robot andador [4]

2.1.3. Robots colaborativos

Además de las clasificaciones descritas anteriormente, en la actualidad va tomando fuerza en el mercado una categoría de robots denominada robots colaborativos. Son robots diseñados para realizar tareas en colaboración con los trabajadores humanos y por tanto están diseñados para garantizar la seguridad del trabajador cuando entra en contacto directo con el robot. Esto quiere decir que están construidos con materiales ligeros, contornos redondo y sensores en la base del robot o en las articulaciones que miden y controlan la fuerza y la velocidad y se aseguran de que no excedan los umbrales definidos en caso de que se produzca contacto con la persona [8].

En un entorno colaborativo, una persona aporta destreza, flexibilidad y capacidad de resolver los problemas, mientras que un robot colaborativo tiene fuerza, resistencia y precisión para realizar una determinada tarea.

En este proyecto se utiliza un robot colaborativo que tiene dos brazos manipuladores; el YuMi de ABB.

YuMi es el primer robot del mundo realmente colaborador, capaz de trabajar mano a mano en las mismas tareas que el ser humano al mismo tiempo. Permite manipular todo tipo de objetos y con un nivel de precisión bastante elevado [9].

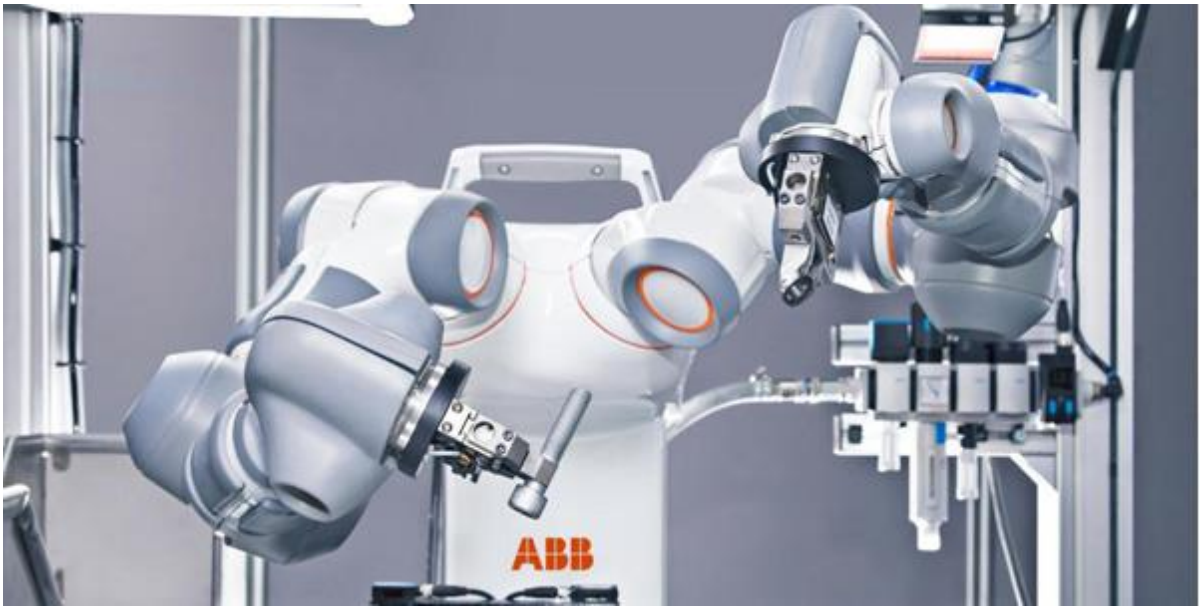


Fig.2.5 Robot colaborativo YuMi [5]

2.2. SISTEMAS OPERATIVOS ROBÓTICOS

La mayoría de los robots, en la actualidad funcionan utilizando su propio sistema operativo. De una forma parecida al funcionamiento de un ordenador común, el robot utiliza un sistema operativo (también conocido como framework) que permite al usuario utilizar el hardware del robot, controlar el dispositivo a bajo nivel, comunicar varios procesos y usar paquetes [21].

Utilizar un sistema operativo propio en robots permite ahorrar tiempo en la implementación de algoritmos ya que no hace falta tener muchos conocimientos acerca del robot para poder programarlo. Además, el código de los programas que se realizan se puede utilizar en cualquier otro robot que funcione con el mismo sistema operativo.

Pero utilizar un sistema operativo propio también presenta ciertas desventajas. Un ejemplo de ello es que en algunos casos sea posible implementar una aplicación en un robot y cuando se quiera hacer la misma aplicación en otro, el sistema operativo no lo permita ya que no están disponibles las mismas funciones para ser utilizadas. Otra desventaja es la incompatibilidad de los SO entre ellos, cosa que limita la posibilidad de comunicar dos sistemas robóticos. Por último, se podría mencionar que en el caso de utilizar varias marcas de robots con varios sistemas operativos diferentes se debe aprender a utilizar varios programas, esto conlleva a la necesidad de dedicar tiempo al aprendizaje de cada una de las marcas de los robots, cosa que supone una pérdida económica.

Los sistemas operativos robóticos se pueden clasificar de la siguiente forma [13]:

2.2.1. Sistemas Operativos Propios

Gran parte de fabricantes de robots crean sus dispositivos para que trabajen utilizando un sistema operativo que ellos mismos crean. De esta forma se consigue optimizar la programación y el funcionamiento de las máquinas ya que este software suele estar ya integrado en el robot. Además, se evitan problemas de compatibilidad entre los robots de la misma marca pudiéndose de esta forma intercambiar información y código entre los diferentes modelos.

Algunos de los ejemplos más importantes de empresas de robots y de sus sistemas operativos son:

- ABB: Se trata de una multinacional suiza especializada en automatización y robótica industrial. Sus robots utilizan un lenguaje de programación de alto nivel: RAPID. Las características más importantes de este lenguaje son la multitarea, la programación modular o las distintas rutinas como por ejemplo las interrupciones. Con el fin de hacer más cómoda la programación del robot, ABB proporciona un simulador propio llamado ROBOTSTUDIO. Este lenguaje de programación y este simulador se han utilizado para la realización de este proyecto ya que los robots que se utilizan para el desarrollo del trabajo es el robot Yumi y el IRB 140 de la empresa ABB.

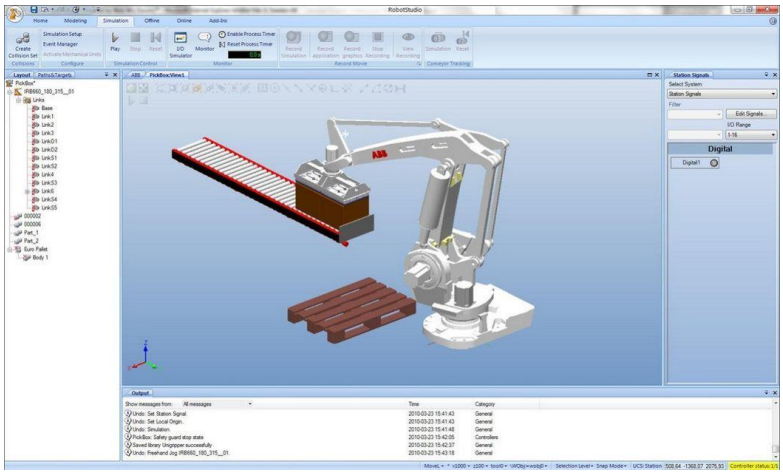


Fig.2.6 RobotStudio [6]

- KUKA: Se trata de otra multinacional especializada en robótica industrial. Sus robots generalmente se utilizan para las líneas de producción de las empresas automovilísticas. El software propio que desarrollan para sus robots en el “Kuka System Sofware”.
- FANUC: Se trata de una empresa japonesa encargada también de la robótica industrial. Está especializada en automatización en el campo de la robótica y el control numérico. De la misma forma que los ejemplos citados anteriormente, esta empresa utilizas su propio software y proporciona un simulador llamado ROBOGUIDE.

Este tipo de software se suele utilizar en el ámbito industrial ya que, en el ámbito de la investigación, por ejemplo, interesa más utilizar herramientas de código abierto para que de esta forma la programación realizada se pueda utilizar en varios dispositivos.

2.2.2. Arquitecturas Orientadas a Servicio

Con el fin de crear sistemas operativos versátiles y de código abierto, los desarrolladores emplean herramientas que descomponen los diferentes procesos de los sistemas operativos y obtienen un bajo nivel de acoplamiento.

Un ejemplo sería la arquitectura orientada a servicios (Service Oriented Architecture). Este tipo de arquitectura se define de la siguiente forma [22]:

"Estilo resultante de políticas, prácticas y frameworks que permiten que la funcionalidad de una aplicación se pueda proveer y consumir como conjuntos de servicios, con una granularidad relevante para el consumidor. Los servicios pueden invocarse, publicarse y descubrirse y están abstraídos de su implementación utilizando una sola forma estándar de interface".

Los elementos más importantes que forman parte de la red SOA son los nodos de la red, que pueden actuar como proveedores o como consumidores, los servicios y la orquestación, que es la parte que se encarga de la secuenciación y la organización del sistema.

En la red SOA, los nodos que forman parte de ella pueden hacer uso de los servicios y la información que otros nodos de red les proporcionan, sin importar la manera de ejecutar los servicios o de publicar la información. Esto quiere decir que los servicios son autónomos y que los nodos trabajan como proveedores de servicios o consumidores débilmente acoplados. Un ejemplo de ellos, sería el hecho de que un servicio programado en C podría ser utilizado por una aplicación programada en Java. Esta arquitectura se basa en estándares como por ejemplo XML, HTTP, SOAP, etc. [23]

Los servicios del SOA deben tener una descripción de cómo se utiliza, además de garantizar que se puede utilizar en distintas aplicaciones y servicios. Finalmente, su uso debe ser independiente de otros servicios.

La ventaja más importante de esta arquitectura es que si se quiere implementar una aplicación que necesite alguna tarea ya desarrollada en otras aplicaciones, no es necesario volver a construir esta tarea, sino que únicamente se deberá llamar a los servicios que ya existen y utilizarlos.

2.2.3. ROS: Robot Operating System

Según la página oficial [24], se puede definir ROS como un framework flexible para el desarrollo del software para robots. Es un conjunto de herramientas, librerías y convenciones cuyo objetivo final es hacer más sencilla la tarea de crear un comportamiento complejo y robusto en las distintas plataformas robóticas.

Se trata de una plataforma de código abierto con licencia BSD donde el trabajo que es desarrollado por unos equipos puede ser utilizado por otro grupo que necesita funcionalidades similares. De esta forma los desarrolladores se centran en un trabajo que nadie antes ha descubierto ahorrándose de solucionar problemas que otros ya han conseguido solucionar.

ROS es soportado de forma oficial por C++ y Python ya que C++ es el lenguaje más extendido y utilizado en la actualidad y Python presenta una gran sencillez y potencia. También soporta lenguajes como Lisp, Lua o Java, aunque todavía está en fase de desarrollo.

2.3. VISIÓN ARTIFICIAL

Para obtener una buena interacción del robot con el entorno hace falta desarrollar sistemas avanzados de inteligencia artificial y por tanto se deben utilizar nuevas técnicas en donde además de la utilización de los sensores, se usen otro tipo de elementos más complejos. Uno de estos elementos puede ser la visión artificial.

Fig.2.7 Visión artificial [7]

El funcionamiento de la visión artificial se podría describir de la siguiente forma: utilizando una cámara el programa desarrollado es capaz de capturar imágenes y entender la escena que está capturando, con esto el programa puede sacar datos de la escena que pueden ser utilizados de igual forma que los datos obtenidos por los sensores tradicionales. Se le llama visión artificial ya que obtiene datos a partir de lo que la cámara puede "ver", igual que lo hace la visión humana.

Los pasos que se sigue en la visión artificial son los siguientes [1]:

- Captura de la imagen.
- Procesamiento y digitalización de la imagen.
- Análisis de la imagen.
- Obtención de resultados respecto a la imagen tomada de la escena real.

Aunque la visión artificial se utiliza en muchos ámbitos, estos son los que más la utilizan [1]:

- Automoción: Inspección de la fabricación y ene el ensamblaje de las piezas.
- Alimentación: Control de calidad de los productos alimentarios.

- Envases y ensamblajes: Controla la presencia y ausencia de marcadores.
- Electrónica: Controla la correcta soldadura y ensamblaje de las piezas.
- Logística e identificación

Cada vez más, la visión artificial adquiere importancia ya que presenta ventajas bastante interesantes:

- Aumento de la productividad.
- Disminución de la pérdida de materiales.
- Reducción de costes.
- Producto final de mejor calidad.

En los últimos años esta tecnología ha ido adquiriendo importancia y para simplificar la utilización de la visión artificial, se han creado librerías y programas específicas para este campo. De esta manera, no se debe tener muchos conocimientos de programación ni de visión artificial para poder desarrollar una aplicación. Las librerías y los programas presentan funciones específicas de visión y se pueden utilizar con facilidad. Son muchas las librerías que pueden utilizarse para desarrollar aplicaciones, las más importantes son las siguientes [2]:

- Torch3vision: Escrita en C++ y dispone de procesamiento básico de imágenes, algoritmos de extracción de características y detección de caras utilizando *Haar-like features*. Es libre y tiene licencia BDS.

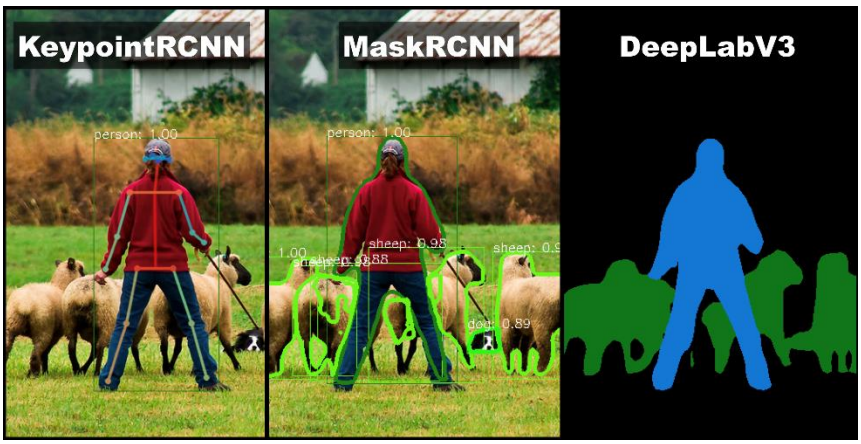


Fig.2.8 Ejemplo de Torch3vision [8]

- VLX: También escrita en C++, tiene la mayoría de los algoritmos más utilizados en visión. Se trata de un conjunto de librerías que ofrecen una muy completa funcionalidad. Dispones de características bastante interesantes y una de ellas es la posibilidad de usar únicamente las

librerías que nos resulten útiles ya que no hay dependencias entre ellas. Las librerías que forman estos conjuntos son las siguientes:

- VNL (*Numerics*)
- VIL (*Imaging*)
- VGL (*Geometry*)
- VSL (*Streaming I/O*)
- VBL (*Basic templates*)
- VUL (*Utilities*)

- RAVL: También escrita en C++, proporciona los elementos básicos de una librería de visión artificial. Los elementos diferenciadores son el soporte para herramientas de audio o interfaces de usuario basadas en GTK.

- LTI-lib: Dispone de más de 500 clases, entre las que se incluyen para álgebra lineal, clasificación, procesamiento de imágenes.

- OpenCV: Es la librería más conocida. Incluye funciones de propósito general para procesamiento de imágenes, descripciones geométricas, segmentación, etc. La característica añadida es la posibilidad de utilizar capacidades de computación de las GPU. Además, permite el uso de las librerías de Intel que incluyen una larga lista de funciones optimizadas para procesadores Intel. Disponible para Linux, Windows y Android. Se puede programar en Java, Python C y C++.

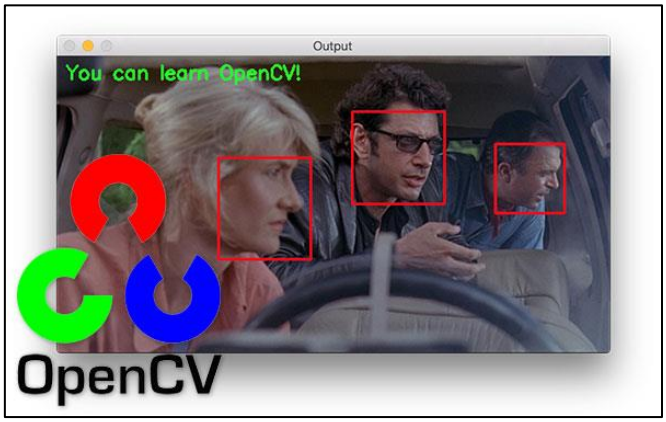


Fig.2.9 Ejemplo de programa OpenCV [9]

OpenCV es la librería que se ha utilizado para realizar este proyecto. Se ha elegido esta librería ya que ofrece muchas posibilidades y funciones a la hora de realizar una aplicación. También se ha tenido en cuenta el hecho de que es la librería más conocida para aplicaciones de visión artificial y por tanto se puede tener fácil acceso a la información acerca de las funciones que ofrece.

2.3.1. Realidad aumentada

La realidad aumentada es el término utilizado para describir el conjunto de tecnologías que permiten al usuario visualizar parte del mundo real en un dispositivo tecnológico con información gráfica añadida. Los dispositivos añaden información virtual a la información física ya existente. Los elementos físicos tangibles se combinan con elementos virtuales creando de esta forma lo que se conoce como realidad aumentada en tiempo real. La realidad aumentada toma como base la realidad física y añade capas de información digital.

Ronald Azuma decía que la realidad aumentada tenía las siguientes características [3]:

- Combina elementos reales y virtuales.
- Es interactiva en tiempo real.
- Esta registrada en 3D.

Fig.2.10 Ejemplo de realidad aumentada [10]

Si se observa la escala de continuidad de la realidad "Milgram-Virtauality Continuum" se puede encontrar que la realidad aumentada está más cerca del entorno real mientras que la virtualidad aumentada está más cerca del entorno virtual [3].

Fig.2.11 Escala de continuidad de la realidad [11]

Muchas veces se confunden los términos realidad aumentada, realidad virtual y realidad mixta. Para evitar esta confusión se debe tener en cuenta los siguientes aspectos [3]:

- La realidad virtual construye un mundo nuevo, mientras que en la realidad aumentada se produce un entorno real y además se añaden elementos virtuales a este entorno. La realidad mixta es una mezcla entre la realidad virtual y la realidad aumentada que ofrece la posibilidad de crear espacios donde interactúan tanto elementos reales como virtuales.

- La realidad virtual y la realidad mixta necesitan de un elemento aportado por el individuo, como por ejemplo unas gafas, mientras que en el caso de la realidad aumentada únicamente basta con una aplicación en el móvil o Tablet.

Fig.2.12 Realidad virtual [12]

Para conseguir crear una realidad aumentada se necesitan ciertos elementos [3]:

- Cámara: Capta la imagen del mundo real.
- Procesador: Elemento hardware que combina la imagen con la información que debe sobreponer.
- Marcador: Se encarga de reproducir las imágenes creadas por el procesador y donde se puede ver el modelo en 3D.
- Software: Gestiona el proceso.
- Pantalla: Muestra la combinación de los elementos virtuales y los reales.
- Conexión a internet: la información del entorno es enviada al servidor remoto y se recoge la información virtual que se ha construido.
- Activador: Elemento que el programa utiliza con el fin de reconocer el entorno real y seleccionar la información virtual que se debe añadir.

2.3.1.1. Tipos de realidad aumentada

Dentro de la realidad aumentada existen diferentes grados que se clasifican según la complejidad de las aplicaciones de la realidad aumentada [3]:

- Nivel 0 (Enlazado con el mundo físico): En este nivel no se puede añadir información 3D a la realidad del mundo físico. Únicamente se utilizan elementos 1D, como por ejemplo el código de barras, o elementos 2D, como es el caso de los códigos QR. Estos elementos solo sirven como hiperenlaces a otros contenidos.

- Nivel 1 (Realidad con Marcadores): Los marcadores son utilizados para reconocimientos de patrones 2D. Una vez realizado el reconocimiento se integran elementos tridimensionales asociados con cada marcador. Esto hace posible dibujar una animación 3D sobre el marcador detectado.

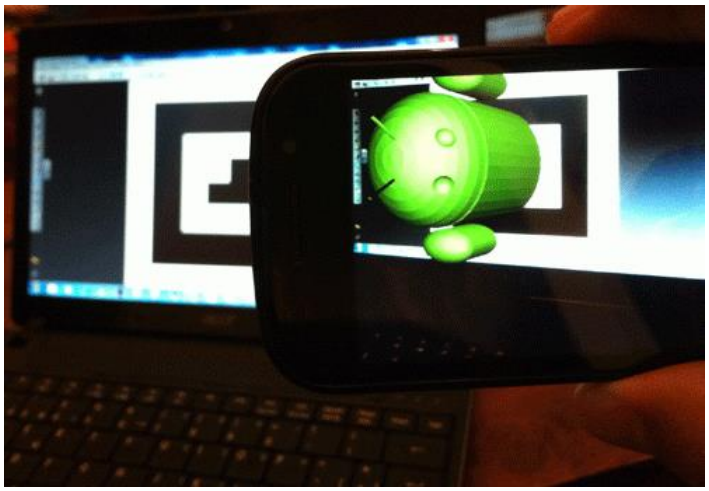


Fig.2.13 Realidad aumentada con marcadores [13]

- Nivel 2 (Realidad sin Marcadores): Se puede prescindir de marcadores físicos. Cuenta con el reconocimiento de superficies, donde el dispositivo es capaz de detectar, en tiempo real, una superficie en el entorno y añadir el contenido virtual a dicha superficie.

- Nivel 3 (Visión aumentada): Es el nivel más alto de complejidad y todavía está en proceso de desarrollo. Lo que se pretende es ofrecer al usuario una experiencia mucho más personal e inmersiva dándole la capacidad de interactuar con el entorno y otras aplicaciones.

En el proyecto que se expone en este trabajo se utilizarán aplicaciones de la realidad aumentada de nivel 1 ya que son los que ofrecen la posibilidad de reconocer marcadores y obtener información acerca de su localización en el mundo real.

2.3.1.2. *Software para la realidad aumentada*

En la actualidad existen multitud de software para la realidad aumentada, para todo tipo de dispositivos y para todos los sistemas operativos. Estas librerías son una buena opción para realizar cualquier trabajo de realidad aumentada ya que no hace falta empezar desde cero para desarrollar la aplicación. Ya tienen creadas algunas funciones que se pueden utilizar simplemente integrando la librería en el proyecto.

A continuación, se presentarán algunos de los ejemplos de software libre para realidad aumentada:

- ARToolkit [4]: Utiliza las capacidades de seguimiento de vídeo, para calcular, en tiempo real, la posición de la cámara y la orientación relativa a la posición de los marcadores físicos. Consigue resolver dos principales problemas de la realidad virtual que son el seguimiento de punto de vista y la interacción objeto virtual. Una vez que se sabe la posición de la cámara real, la cámara virtual se sitúa en el mismo punto y los modelos 3D se sobreponen sobre el marcador real.

- ARTag: Es una librería que funciona de manera similar a la anterior pero que a diferencia de esta puede usar procesos de tratamientos de imagen más complejos y desarrollar aplicaciones con mayor inmunidad lumínica. Presenta dos modos de funcionamiento: “Magic Mirror” y “Magic Lens”.

- ARUco: También es una librería de código abierto basada en OpenCV fácil de usar y permite desarrollo de aplicaciones a partir de los diversos ejemplos que tiene programados.

- ARToolKitPlus: Se trata de una llibrería orientada a programadores avanzados ya que no tiene funciones definidas que capturen imágenes con una cámara o dibujen elementos 3D.

- ATOMIC Authoring Tool: Esta especialmente diseñada para no-programadores. Ofrece una interfaz que le permite al usuario entender el funcionamiento sin necesidad de que tenga muchos conocimientos. Es multiplataforma lo que permite que se pueda utilizar en los sistemas operativos más conocidos.

En este proyecto se ha elegido utilizar la librería ARUco debido a su sencillez y eficacia a la hora de programar nuevas aplicaciones.

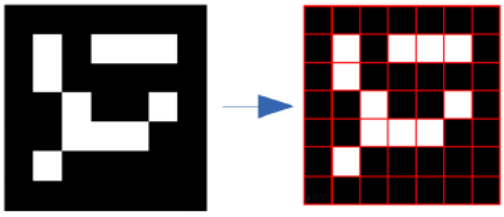


Fig.2.14 Marcadores ARUco y su codificación [14]

2.3.1.3. Funcionamiento de la realidad aumentada

Si se desea desarrollar un proyecto de realidad aumentada se deben tener en cuenta las siguientes consideraciones:

- Debe tener un dispositivo capaz de capturar imágenes ya sea haciendo vídeo o solo fotografías. En el caso de este proyecto se ha utilizado una cámara Logitech 2D y una cámara 3D.

- Tener un hardware donde se tenga instalado el software de realidad aumentada necesario para que se pueda ejecutar la aplicación desarrollada. En este caso se utiliza un portátil.

- Con el fin de poder observar cómo se construye el mundo virtual sobrepuesto al mundo real, se debe tener un display. En este caso se utiliza la pantalla del ordenador donde se muestra en una ventana lo que la cámara captura del mundo real y en otra ventana los elementos virtuales añadidos a los que se captura del mundo real

- En último lugar es necesario tener marcadores. El marcador debe ser lo suficientemente grande como para ser detectado por la cámara según el campo de trabajo de la aplicación y lo suficientemente pequeño como para que dentro de esa área de trabajo se puedan realizar los desplazamientos necesarios para la aplicación.

Con respecto al funcionamiento de las librerías de realidad aumentada, se puede decir que cada una de ellas está programada de una forma diferente y sus funciones difieren. Pero la forma trabajar utilizando los marcadores se podría generalizar de la siguiente manera [15]:

- En primer lugar, un dispositivo captura las imágenes o vídeo del mundo real y las envía a un dispositivo que es el que se encarga de procesar la información.

- El siguiente paso es el procesamiento de la imagen dentro de la cual se debe encontrar un marcador.

- Cuando el marcador ha sido detectado la librería procesa las funciones que tiene implementadas y “saca” la posición y orientación del marcador siento el origen de coordenadas del sistema el punto central de la cámara.

- Finalmente, sobre los marcadores detectados se dibuja el mundo virtual. En el caso de la aplicación desarrollada en este proyecto, encima del punto central de los marcadores se dibujan los ejes (x, y, z) cada uno en un color diferente.

2.3.1.4. Marcadores ARUco

Como ya se había comentado antes ARUco es una librería de código libre desarrollada por investigadores de la Universidad de Córdoba que se utiliza para la detección de marcadores cuadrados en las imágenes. Utilizando los parámetros de la calibración de la cámara nos brinda la posibilidad de saber la posición y orientación del marcador respecto la cámara o viceversa en coordenadas del espacio mundo.

Esta librería está escrita en C++ y para poder utilizar la requiere de la instalación de la librería de OpenCv. ARUco es capaz de detectar también marcadores de otras librerías, como, por ejemplo, ArToolKit+, Chilitags, AprilTags.

Los marcadores ARUco presentan un exterior negro en cuyo interior se encuentra una matriz binaria codificada. Esta matriz es única para cada marcador y gracias a esto se pueden detectar marcadores. En función del diccionario utilizado, los marcadores tienen más o menos bits. Si el número de bits es muy grande mayor será el diccionario y por tanto menos será la posibilidad de confusión entre los marcadores. Para una correcta detección de los marcadores la resolución de la imagen adquirida por la cámara debe ser mayor. Para que el programa pueda detectar el marcador, también se debe indicar el tamaño de este en el mundo real.

El proceso de detección de marcadores se podría resumir en los siguientes pasos [15]:

- Para la obtención de bordes, se aplica un filtro de umbral adaptativo a la imagen. Una vez obtenidos los bordes se marcan los cuadrados que indicarán si es posible que se trate de un marcador o no. A los candidatos se les aplica nuevamente un segundo filtro para eliminar los que son demasiado pequeños o los que están muy próximos.

- A continuación, se pasa a detectar la matriz de bits. Para poder realizar este paso, a cada candidato se le aplica una transformación de perspectiva que permite normalizar su forma. Posteriormente, mediante el método de Otsu, se les aplicará a los candidatos otro filtro de valor umbral con el fin de separar los bits negros de los blancos. El marcador es separado en celdas en función del tamaño del borde del marcador y para determinar si es blanco o negro se analiza el centro de cada celda. Como último paso se analiza lo obtenido para determinar si existe algún diccionario al cual se corresponde y de esta forma poder sacar las medidas necesarias,

Fig.2.15 Corrección de perspectiva [15]

Fig.2.16 División en celdas [15]

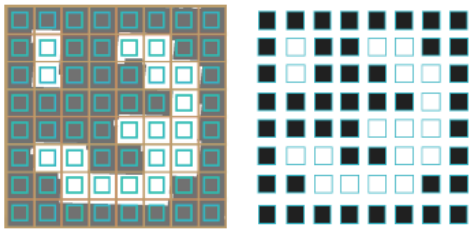


Fig.2.17 Extracción de bits del marcador [15]

2.3.2. CÁMARAS

Uno de los dispositivos imprescindibles en el campo de la visión artificial es la cámara. Se puede definir la cámara como un dispositivo utilizado para capturar imágenes estáticas o en movimiento. Las cámaras actuales se combinan con elementos sensibles al espectro visible y su uso principal es capturar la imagen que se encuentra en el campo visual.

Las cámaras fotográficas tienen una cámara oscura cerrada, con una abertura en uno d ellos extremos para que pueda entrar la luz, y una superficie plana de formación de la imagen para capturar la luz en el otro extremo

2.3.2.1. Tipos

En la actualidad existe una gran variedad de cámaras, entre las más importantes se encuentran las siguientes [5]:

- Cámaras compactas: Fáciles de usar y no requiere de conocimientos para ser usada. Son las de menor coste, con visor óptico directo y objetivo no intercambiable.

Fig.2.17 Cámara compacta [14]

- Cámaras Puentes: Tienen las mismas características que las compactas, pero a diferencia de estas, las cámaras puentes tienen esteroide. Son las cámaras que se usan para los momentos de la vida cotidiana.

- Cámaras instantáneas: Al tomar la foto la saca directamente en papel fotográfico, aunque algunas tienen la posibilidad de guardar la fotografía en una tarjeta de memoria.

Fig.2.18 Cámara instantánea [15]

- Cámaras Mirrorless: Llevan un visor electrónico donde se puede ver la imagen y la exposición "en directo". No llevan espejo, con lo que se reduce el tamaño del cuerpo y la lente está más cerca del sensor.

- Cámara digital: Dispositivo electrónico capaz de capturar y almacenar fotografías electrónicamente.

- Cámara estereoscópica: Presenta capacidad para capturar imágenes en tres dimensiones. La visión humana, produce dos imágenes (uno por cada ojo) que posteriormente en el cerebro se mezclan y se crea la imagen 3D. Estas cámaras imitan este comportamiento y utilizando dos objetivos captan la imagen en el mismo instante y obtienen las fotografías en 3D.

Fig.2.19 Cámara 3D [16]

2.3.2.2. Elementos de una cámara

Como componentes básicos de un cámara podemos encontrar los siguientes [7]:

- Elemento fotosensible: se trata de un elemento sensible a la luz que registra la imagen que procede del objetivo. Normalmente se utiliza una película fotográfica, papel fotográfico auto revelable o un sensor de imagen electrónico.

- Visor: Sistema óptico cuya función es permitir encuadrar el campo visual que abarca la fotografía. Sirve al fotógrafo previsualizar la relación motivo/entorno que abarca el objetivo.

Fig.2.20 Visor de una cámara [17]

- Enfoque: mecanismo que ofrece la posibilidad de enfocar a distancias cortas y de enfocar únicamente ciertos planos de una escena para destacarlos.

- Diafragma: regula la apertura del sistema óptico. Normalmente es un disco puesto en el objetivo de la cámara que permite restringir el paso de la luz.

Fig.2.21 Diafragma de una cámara [18]

2.3.2.3. Calibración de cámaras

Como ya sabemos las imágenes capturadas por una cámara tienen cierta distorsión y esto se debe al hecho de que la lente y el plano de la imagen no son exactamente paralelos. La calibración de la cámara se encarga de corregir este error. El primer paso imprescindible a la hora de trabajar con visión artificial es la calibración de la cámara.

Se puede distinguir entre la calibración de color y la calibración geométrica. La calibración de color permite que los colores de las imágenes adquiridas se aproximen lo máximo posible a los colores de la vida real. Por otro lado, la calibración geométrica se utiliza para corregir distorsiones que se producen cuando hacemos una representación 2D del mundo real que es 3D. En este proyecto únicamente se utiliza la calibración geométrica por tanto a continuación se explicará este tipo de calibración.

La calibración de la cámara permite obtener los valores intrínsecos y extrínsecos de esta y así existe la posibilidad de reconstruir el entorno y poder obtener medidas correctas del mundo tridimensional a partir de fotografías de dos dimensiones.

Se les denomina parámetros intrínsecos a aquellos correspondientes a las características internas de la cámara: dimensiones de los píxeles (α x y α y), ángulo de inclinación en el eje de la imagen (γ) posición del centro de proyección C (u_0, v_0) y longitud focal.

Los parámetros extrínsecos son aquellos que representan la orientación y localización de la cámara respecto a un sistema de coordenadas y por tanto se puede obtener la relación que existe en las coordenadas de los puntos de la imagen (2D) y las coordenadas de los puntos del mundo real (3D).

Cuando se calibra una cámara lo que se hace es capturar varias imágenes con un patrón conocido de calibración y se obtienen un conjunto de ecuaciones que relacionan los puntos del patrón en el mundo real con los puntos de ese patrón en la imagen. Matemáticamente, la anterior explicación se podría expresar de la siguiente forma:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = A \begin{bmatrix} R & T \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Donde $[u\ v\ 1]^T$ son las coordenadas en píxeles de la posición del punto y $[\ x\ y\ z\ 1]^T$ son las coordenadas de la posición de ese punto en el mundo real. A es la matriz intrínseca y se expresa de la siguiente forma:

$$A = \begin{bmatrix} \alpha_x & \gamma & u_0 \\ 0 & \alpha_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

R y T son los parámetros extrínsecos; la R corresponde a la rotación y la T corresponde a la translación.

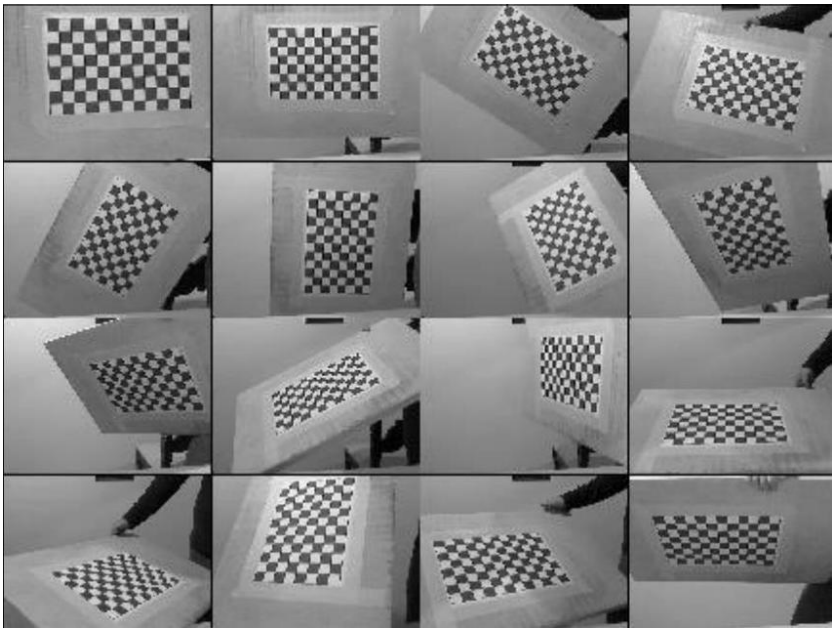


Fig.2.22 Posiciones para calibrar la cámara [19]

2.4. PROTOCOLOS DE COMUNICACIÓN

Con el fin de alcanzar el objetivo de este proyecto es necesario poder comunicar la aplicación de visión artificial con la que controla los movimientos del robot.

Para ello se debe crear un canal de comunicación entre la máquina que hace de cliente (en nuestro caso el ordenador) y la máquina servidor (en este caso el robot). Esto hace imprescindible el uso de algún protocolo de transporte. Los dos protocolos más importantes son los siguientes [14]:

- Protocolo UDP (User Datagram Protocol): es un protocolo basado en el intercambio de datagramas. Permite el envío de estos datagramas a través de la red sin que anteriormente se haya establecido una conexión. Es considerado un protocolo no muy seguro ya que no tiene confirmación ni control de flujo y por este motivo no se ofrece ninguna garantía de que los paquetes lleguen en el mismo orden en el cual fueron enviados. Además, ni siquiera garantiza la llegada de estos paquetes. Se utiliza en la transmisión de voz o vídeo ya que en este caso es más importante transmitir con velocidad y no es necesario garantizar la llegada de absolutamente todos lo bytes. Su principal ventaja es que provoca poca carga adicional en la red ya que además de ser simple también utiliza cabeceras muy simples.

- Protocolo TCP (Transmission Control Protocol): es fundamental en Internet. Se trata de un protocolo fiable ya que posee mecanismos que garantizan la integridad y la llegada en orden de los mensajes. Al tener estas garantías el protocolo utiliza comprobaciones adicionales y como consecuencia es menos eficiente que el UDP, aunque facilita la programación ya que no hay que tener en cuentas los problemas existentes en la red. Algunas de las características de este protocolo son:

1. Orientado a la conexión: dos máquinas se conectan para intercambiar datos. Los sistemas se sincronizan para manejar el flujo de paquetes y adaptarse a la congestión de la red.
2. Operación full-dúplex: la conexión TCP es un par de circuitos virtuales cada uno de los cuales están en una dirección. Únicamente los dos sistemas finales sincronizados pueden usar la conexión.
3. Revisión de errores: existe una técnica denominada checksum que se usa para verificar que los paquetes no estén corruptos.
4. Acuses de recibo: el receptor le transmite una señal al transmisor para indicarle que el paquete enviado ha sido recibido
5. Control de flujo: cuando el transmisor transmite demasiado rápido y llega a desbordar un buffer el receptor descarta paquetes. De esta manera los acuses de que no ha llegado correctamente el paquete llegan al transmisor y esta baja la velocidad de transferencia o deja de transmitir.
6. Servicio de recuperación de paquetes: el receptor puede pedir que le reenvíen un paquete. De igual forma si el paquete no se notifica como recibido, el transmisor lo envía de nuevo.

En este proyecto se utiliza el protocolo TCP debido a la fiabilidad en la transmisión de datos ya que para crear las aplicaciones es importante que se transmitan todos los datos y en el orden en el cual fueron enviados.

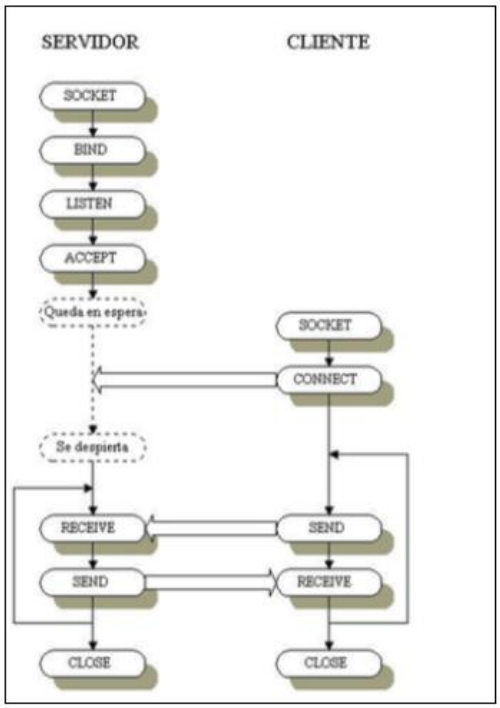


Fig.2.23 Esquema cliente-servidor TCP

2.4.1. Sockets

En el presente trabajo se realizará una aplicación distribuida. La aplicación distribuida se define como aquella que tiene distintos componentes que se ejecutan en entornos separados, normalmente en diferentes plataformas conectadas a través de una red. Existen tres típicas aplicaciones distribuidas: de dos niveles (cliente-servidor), tres niveles (cliente middleware-servidor) y multinivel. Para este trabajo se ha decidido utilizar el modelo cliente servidor.

En este modelo de aplicación distribuida existen dos roles: cliente y servidor. El servidor dispone de recursos necesarios para ofrecer el servicio solicitado por el cliente. Así pues, se puede generar una colaboración entre el cliente y el servidor ya que el cliente solicita al servidor un determinado servicio mientras que el servidor le concede los recursos para la ejecución de dicho servicio. Este tipo de aplicación puede tener el cliente y el servidor en la misma máquina, aunque normalmente esto no ocurre así.

En nuestro proyecto se tiene un robot colaborador que hará de servidor mientras el ordenador donde se ejecuta la aplicación de la detección de marcadores hace de cliente.

Para poder crear las aplicaciones se tendrán que intercambiar datos entre el servidor y el cliente y para ello necesitamos crear un canal de comunicación. Cuando se trata de procesos que están en máquinas distintas la comunicación se lleva a cabo a través de redes de computadoras, que es una tarea muy compleja. Para resolver está complejidad se utiliza el diseño por capas. Cada capa utiliza funciones de la capa inferior y ofrece funciones a la capa superior. Es decir, cada capa recibe de la capa inferior un paquete que está compuesto por control y datos y la envía a la capa superior.

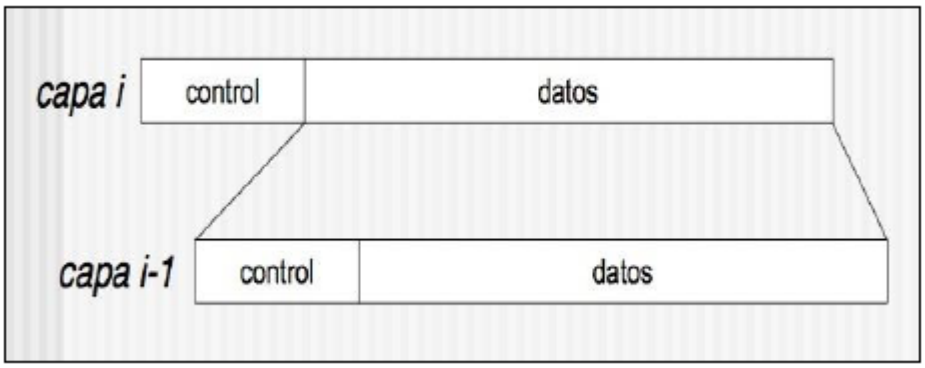


Fig.2.24 Envío de datos de una capa a otra

Uno de los modelos más importantes del diseño por capas es el modelo TCP/IP que está compuesto por cuatro capas diferentes [14]:

- Capa de aplicación: Está compuesta por servicios y aplicaciones de comunicación estándar que puede utilizar todo el mundo. Asegura de que la información se transmita al receptor de un modo comprensible para el sistema, además administra las conexiones y terminaciones entre los sistemas que cooperan. Los protocolos más utilizados en esta capa son: DNS, SMTP, HTTP, FTP.

- Capa de transporte: Administra la transferencia de datos. Garantiza que los datos recibidos sean idénticos a los transmitidos. Los protocolos más utilizados son TCP y UDP.

- Capa de Internet: Se encarga de administrar las direcciones de datos y la entre redes. Los protocolos principales son IP e ICMP.

- Capa de red: Se encarga de la transferencia de datos en el medio de red y de definir las características del hardware de red.

Dentro de esta estructura, los sockets son una interfaz en la capa de transporte.

El socket es un mecanismo de comunicación que permite la comunicación bidireccional entre procesos que estén en la misma máquina o en procesos de máquinas diferentes. Dicho de otra forma, es un punto de comunicación entre dos procesos por el cual se puede emitir o recibir información. La particularidad que presenta este mecanismo de comunicación frente a otros es que posibilitan la comunicación aun cuando ambas máquinas funcionen en distintos sistemas unidos mediante una red. El mecanismo de comunicación vía sockets sigue los siguientes pasos:

- El servidor crea un socket con nombre y espera la conexión

- El cliente crea un socket sin nombre

- El cliente realiza una petición de conexión al servidor

- El cliente realiza la conexión a través de su socket mientras el servidor mantiene su socket original con nombre.

Todo socket viene definido por dos características fundamentales [14]:

1. El tipo de socket: indica la naturaleza y el tipo de comunicación.

2. El dominio de socket: dice dónde se encuentran los procesos que se van a Intercomunicar.

Existen diferentes tipos de sockets, los más destacables son los siguientes:

- SOCK_DGRAM: este tipo de socket no está orientado a conexión y sirve para datagramas. Además, el envío de datagramas es de tamaño limitado, el mensaje se puede perder, duplicar o llegar fuera de secuencia. El protocolo que se utiliza en la capa transporte es el UDP.

- SOCK_STREAM: orientado a conexión, cuando se conecta se hace una búsqueda de un camino libre entre el servidor y el cliente. Este camino se mantiene en toda la conexión. Se utiliza para realizar comunicaciones fiables, de dos vías y con tamaño variable de los mensajes de datos. En la capa de transporte se utiliza el protocolo TCP.

- SOCK_RAW: permiten un acceso a más bajo nivel, pudiendo acceder directamente al protocolo IP del nivel de Red. Tiene un uso más limitado ya que está pensado para desarrollar nuevos protocolos de comunicación o para hacer uso de facilidades ocultas de un protocolo existente.

- SOCK_SEQPACKET: Tiene las características del SOCK_STREAM, pero el tamaño de los mensajes es fijo.

Aunque existen diferentes dominios de comunicación los más empleados son [14]:

- Dominio AF_UNIX: El cliente y el servidor están en la misma máquina. Cada socket tiene una dirección única y en este caso se utiliza como dirección el nombre de un archivo local.

- Dominio AF_INET: El servidor y el cliente pueden estar en cualquier máquina de la red Internet y la comunicación es basada en el conjunto de protocolos TCP/IP. El socket se identifica mediante el par: dirección IP de la máquina, número de puerto.

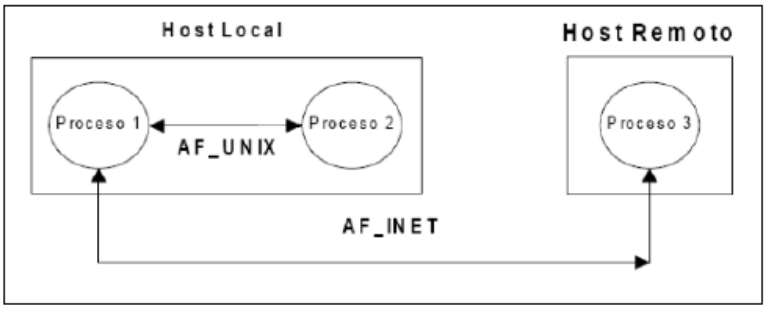


Fig.2.25 Dominio AF_UNIX y AF_INET

3. PARTE PRÁCTICA

En esta parte se explicarán los diferentes pasos realizados en este proyecto hasta conseguir el objetivo final.

3.1. COMUNICACIÓN MATLAB-ROBOTSTUDIO

Como ya se ha dicho anteriormente uno de los robots utilizados para este proyecto es el robot YuMi de la empresa ABB. Esta empresa pone a disposición de sus usuarios un simulador virtual, el RobotStudio. Este simulador permite programar el robot de la misma forma que se programa el robot real y ver los movimientos en pantalla. Estos movimientos, vistos en pantalla, son los mismos que haría el robot real con ese programa.

En el caso de que se vaya a desarrollar una aplicación nueva con el robot, es recomendable hacerlo primero simulado ya que no se sabe exactamente qué movimientos realizará el robot ni tampoco si reaccionará de la forma que uno se espera y de esta forma se evitará que el robot se haga cualquier daño. Una vez visto en el simulador que el robot funciona según el programador quiera, ya se puede pasar a implementar el programa en el robot real.

Debido a lo descrito anteriormente, al inicio del desarrollo de este proyecto también se ha utilizado el RobotStudio.

El primer paso ha sido la comunicación del programa Matlab con RobotStudio. En este caso, el programa Matlab enviaría las posiciones a las que el robot se deba mover y RobotStudio recibiría estas posiciones y movería el robot en los puntos enviados.

Inicialmente se ha creado un fichero de texto en el cual se escriben ciertos movimientos para enviarlos al robot y que el robot los haga. La diferencia de este robot con los demás robots del mercado, es que presenta dos brazos y es entonces cuando surgen las preguntas: ¿cómo se deben ordenar los datos?, ¿envío en una sola trama toda la información de los dos brazos o envío dos tramas diferentes, una para cada brazo? Estas son las preguntas que se debían responder.

3.1.1. Definición de la trama

Para empezar, lo primero que se ha pensado ha sido enviar en una sola trama toda la información correspondiente a los dos brazos ya que de esta forma se ahorraría tiempo de cómputo respecto a si se enviarían dos tramas diferentes. Se debe tener en cuenta que para separar cada uno de los datos, a lo largo de todo el proyecto se utilizará el ";".

Los datos que se debían enviar al robot eran las tres posiciones (x, y, z) y los cuatro cuaterniones (q1, q2, q3, q4). Teniendo en cuenta que el robot utiliza milímetros como medida para mover sus brazos, se pensó que lo mejor, respecto a las posiciones, era enviar como máximo cuatro números enteros ya que el movimiento humano de los brazos era difícil que llegase a alcanzar más de un metro. En cuanto a los cuaterniones, se pensó que lo mejor era enviarlos con tres decimales ya que era una aproximación

que el robot podría interpretar correctamente y mover los brazos según lo deseado. Para que en la misma trama el robot supiese identificar si la información se corresponde al brazo izquierdo o derecho se ha pensado que antes de enviar la información de cada brazo se enviaría el identificador para decir a qué brazo corresponde la información. En este caso el 1 correspondería al brazo izquierdo y el 0 al brazo derecho. De esta forma la información que se debía escribir en el fichero quedaba de la siguiente forma:

1;xxxx;yyyy;zzzz;q1.q1q1q1;q2.q2q2q2;q3.q3q3q3;q4.q4q4q4;0;xxxx;yyyy;zzzz;q1.q1q1q1;q2.q2q2q2;q3.q3q3q3;q4.q4q4q4;

Según podemos contar, en este formato, la trama que se quiere enviar contiene 84 caracteres mientras que el robot solo puede recibir una cadena de como máximo 80 caracteres. Por este motivo la cadena que se pretendía enviar no tenía el formato correcto y se pasó a la idea de enviar una trama diferente para cada brazo del robot. De esta forma el formato que se quería enviar tendría el siguiente aspecto:

1;xxxx;yyyy;zzzz;q1.q1q1q1;q2.q2q2q2;q3.q3q3q3;q4.q4q4q4;

0;xxxx;yyyy;zzzz;q1.q1q1q1;q2.q2q2q2;q3.q3q3q3;q4.q4q4q4;

1;xxxx;yyyy;zzzz;q1.q1q1q1;q2.q2q2q2;q3.q3q3q3;q4.q4q4q4;

0;xxxx;yyyy;zzzz;q1.q1q1q1;q2.q2q2q2;q3.q3q3q3;q4.q4q4q4;

Fig.3.1 Primera trama

Este formato sí que pudo enviarse ya que no superaba el número máximo de caracteres permitido. Por tanto, se pensó que este debería ser el formato para enviar los datos del fichero al robot.

Los datos probados anteriormente eran datos de posiciones aleatorias que se han elegido al azar. La prueba anterior solo se hizo para tener una idea de cómo debía ser la trama y de asegurarse de que la conexión Matlab-RobotStudio se hacía de una forma correcta, enviando y recibiendo datos conforme se esperaba. Pero en este momento se debía dar un paso más y obtener datos a partir de movimientos reales.

3.1.2. Pruebas de comunicación con posiciones generadas fuera de línea

El robot presenta un determinado campo de trabajo y no puede realizar ciertos movimientos que una persona humana sí que podría hacer por este motivo se han probado posiciones extremas, con el fin de poder determinar hasta dónde llega el alcance del robot.

Para hacer pruebas de alcance se ha utilizado un fichero con datos de posiciones obtenidas a partir de movimientos de brazos de un humano y detectado mediante una cámara 3D cuyo desarrollo y programación quedan fuera del ámbito de este TFM.

El problema que se tenía con las posiciones obtenidas utilizando la aplicación de la cámara 3D era que hubo ciertos puntos que producían un salto grande en el cambio de la posición y estos saltos el robot no sería capaz de darlos ya que en poco tiempo debía realizar cambios bruscos en sus posiciones. Por este motivo, se eliminaron estos cambios bruscos y se crea un nuevo archivo de texto con las posiciones modificadas para que el robot sea capaz de seguirlas. Los cambios bruscos se eliminan calculando las diferencias de posiciones entre dos tiempos consecutivos y en caso de que esta diferencia sea grande, el valor de la posición "actual" se cambia de valor al valor de la posición "anterior".

A continuación, se representarán las posiciones, obtenidas con la aplicación, para los brazos y las posiciones modificadas para estos brazos:

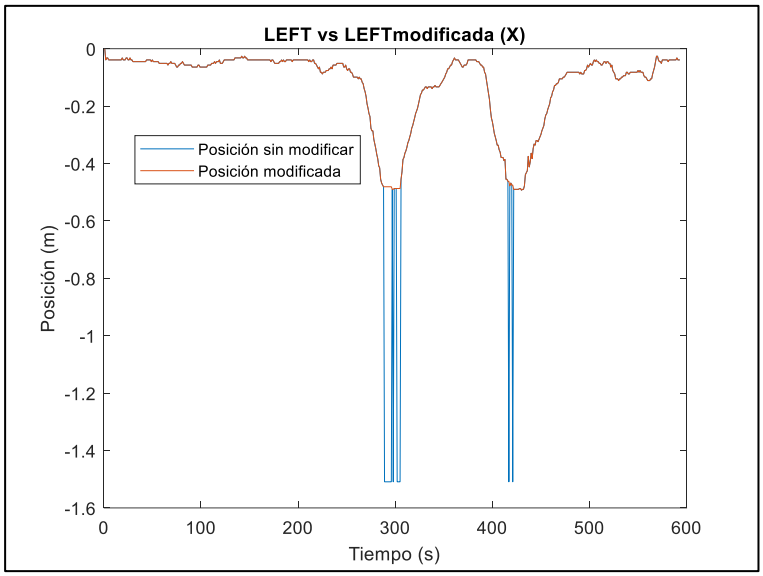


Fig.3.1 La x del brazo izquierdo

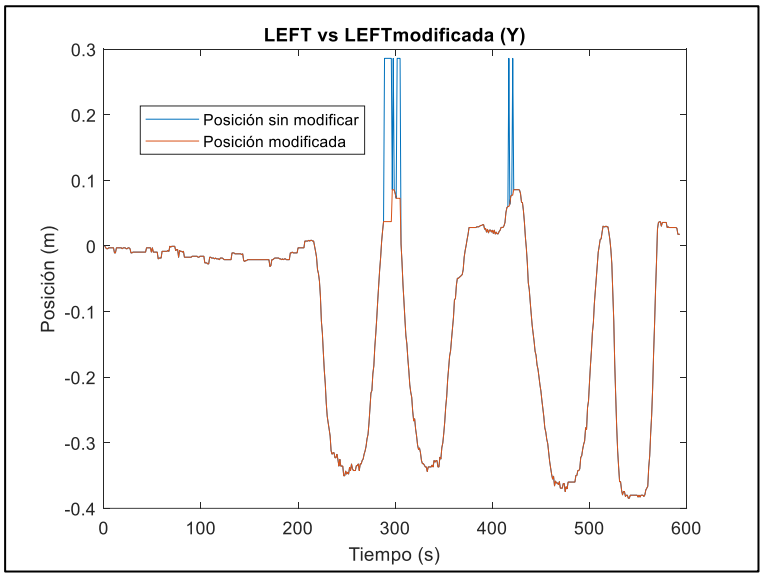


Fig.3.2 La y del brazo izquierdo

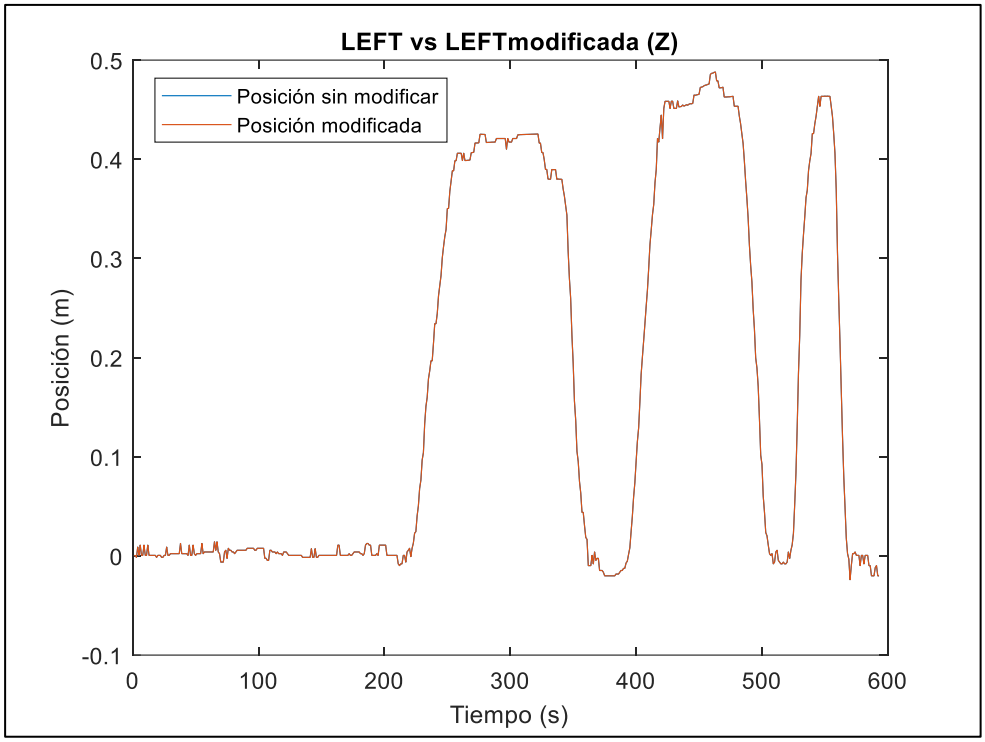


Fig.3.3 La z del brazo izquierdo

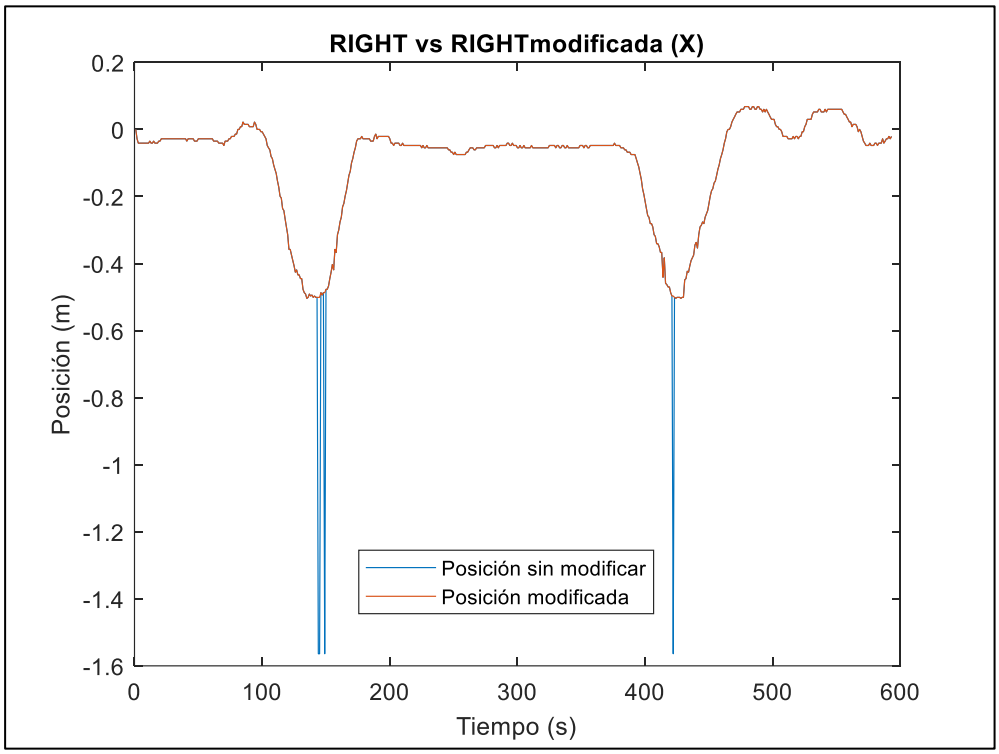


Fig.3.4 La x del brazo derecho

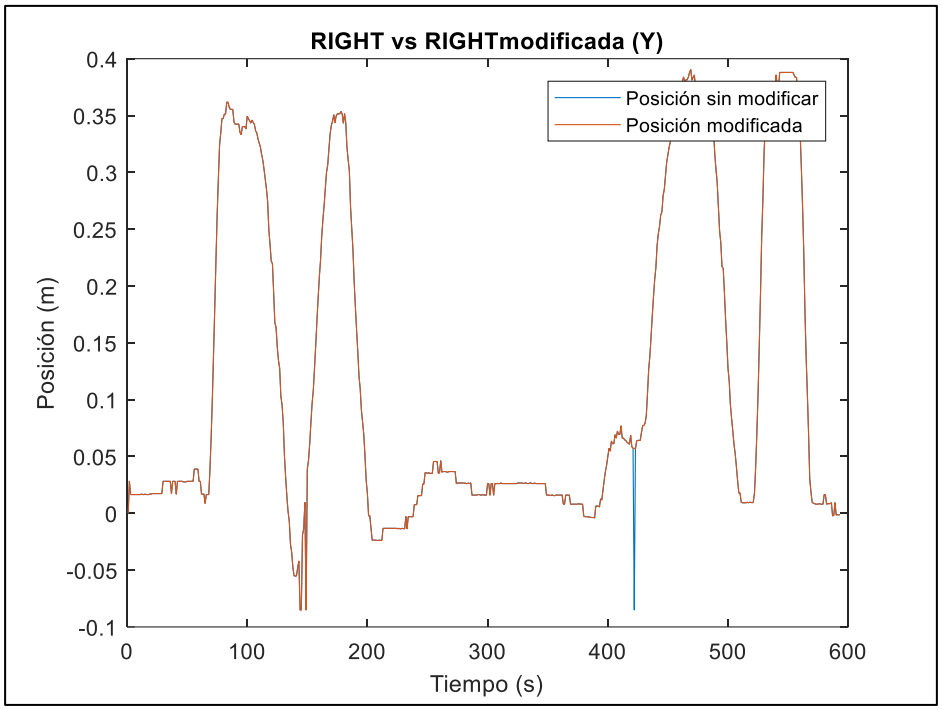


Fig.3.5 La y del brazo derecho

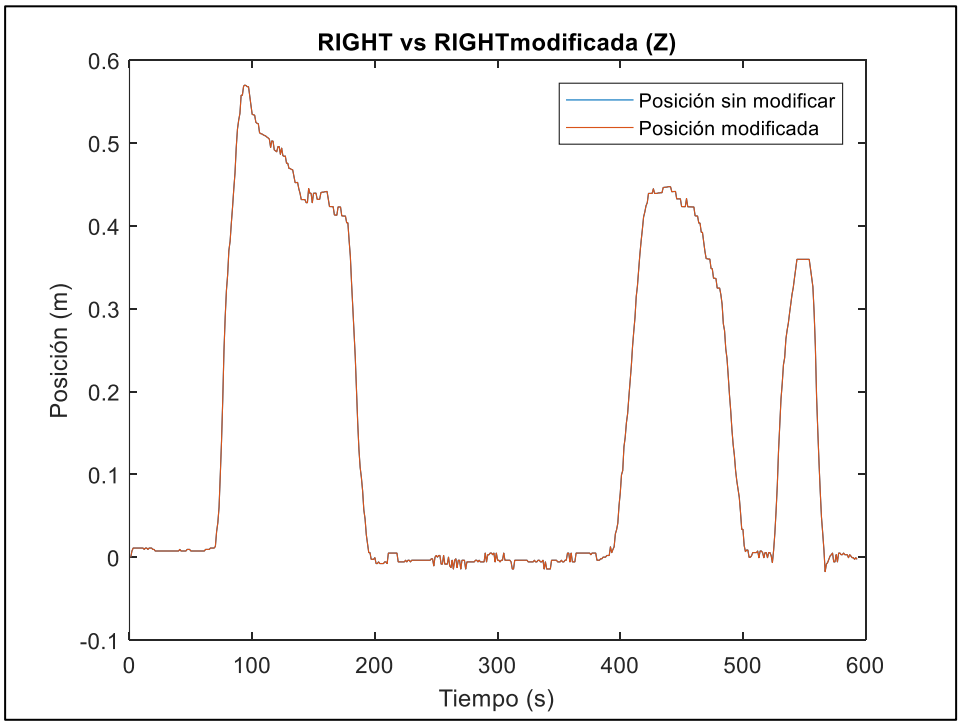


Fig.3.6 La z del brazo derecho

Otra cosa que se debe tener en cuenta a la hora de generar el fichero con los datos para enviar al robot, es que la aplicación de la cámara obtiene la posición en metros y el programa Rapid necesita la posición en milímetros. También se ha observado que los datos obtenidos por la aplicación son en espejo respecto la persona que se está moviendo, como consecuencia de ello, se cambiará el signo de las coordenadas X Y. Utilizando la trama descrita anteriormente, en la que en cada trama se envían los datos correspondientes a uno de los brazos, se han enviado los datos de las posiciones modificadas al RobotStudio para ver cómo se mueve el robot.

Para hacer la comunicación Matlab-RobotStudio lo primero que se hace es abrir la comunicación en RobotStudio como servidor (explicado en el apartado 3.4.1) y a continuación se abre en el Matlab.

Para ello, en el Matlab se utiliza la función `tc_140=tcpip ('127.0.0.1',11000)` donde el primer parámetro indica la dirección IP (en este caso como la comunicación se realiza con el simulador la dirección es el localhost) y el segundo indica el puerto. `Fopen(tc_140)` sirve para abrir la comunicación con las características descritas en la función `tcpip`.

A continuación, en el Matlab se lee el archivo de texto y se crea la cadena que se debe enviar con el formato correspondiente (el descrito anteriormente). Para enviar la cadena se utiliza la función `fwrite (tc_140, strTotal)`. Donde como primer parámetro se le pasa el identificador de la comunicación enviada y como segundo parámetro la cadena que se quiere enviar. Una vez enviada la cadena con las posiciones, en RobotStudio se reciben estas posiciones y se designan al brazo derecho o izquierdo según el identificador que se les ha enviado. Cuando la cadena se recibe, el Robot-Studio envía otra cadena donde indica a Matlab que ya ha recibido los datos y por tanto Matlab puede seguir enviando información. La recepción de datos en Matlab se hace utilizando la función `fread`.

Según se pudo observar en la simulación realizada, había un momento en el que el robot alcanzaba un error cinemático ya que no podía alcanzar el valor de la posición deseada. Esto se debe a que su campo de trabajo no es tan amplio como el de la persona humana que realizó los movimientos. Para solucionar este problema lo que se ha hizo es dividir los valores de las posiciones enviadas al robot. Así pues, si por ejemplo, se divide por el número 2 y la persona se movió 2 cm el robot solo se moverá 1 cm de esta forma cuando la persona extiende del todo su brazo el robot recibirá el dato de la posición de la mano extendida dividida por 2 y de esta forma se reduce el campo de trabajo del robot y se observa entre qué valores de posiciones puede trabajar el robot. Se intentó dividir por el número 2 y 3 pero aun así seguía habiendo errores. Si se dividía por 4 los valores que se el robot recibía estaban dentro de su área de trabajo.

Esto hace que en los programas desarrollados posteriormente siempre se tengan en cuenta las limitaciones y se programe de tal forma que el robot no llegue a alcanzar estos límites.

Otra cosa que también se debió tener en cuenta fue la posición inicial del robot ya que según en qué posición empezaba el robot, conseguía alcanzar una posición más lejana o no. La posición inicial elegida en este caso, con el fin de que pueda alcanzar todas las posiciones, es la que se muestra en la siguiente figura:

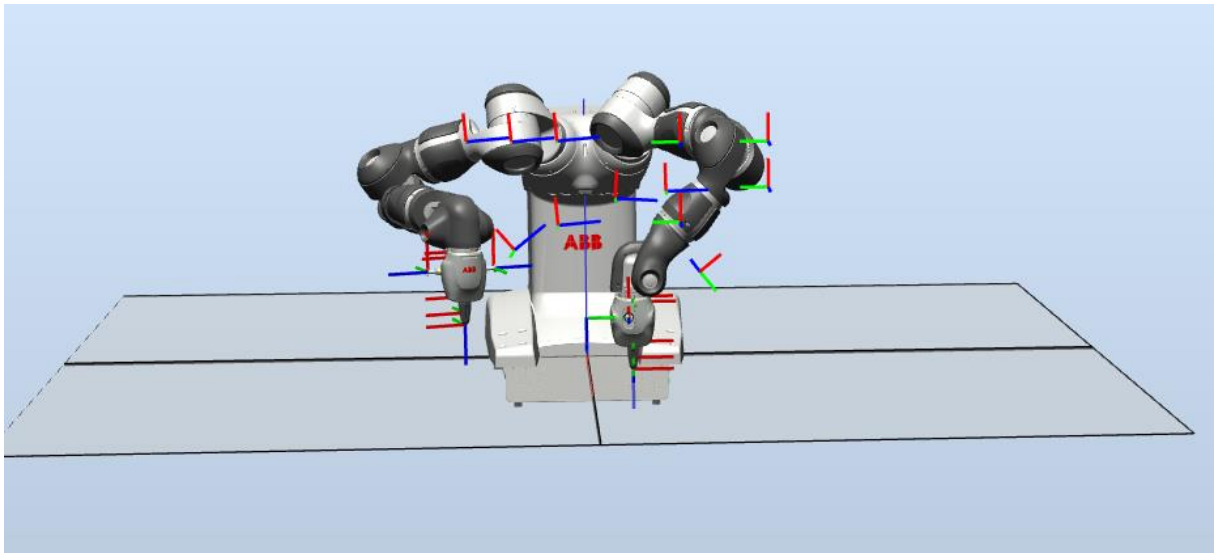


Fig.3.7 Posición inicial del robot

En las siguientes fotos se puede observar cómo imita el robot al comportamiento humano:

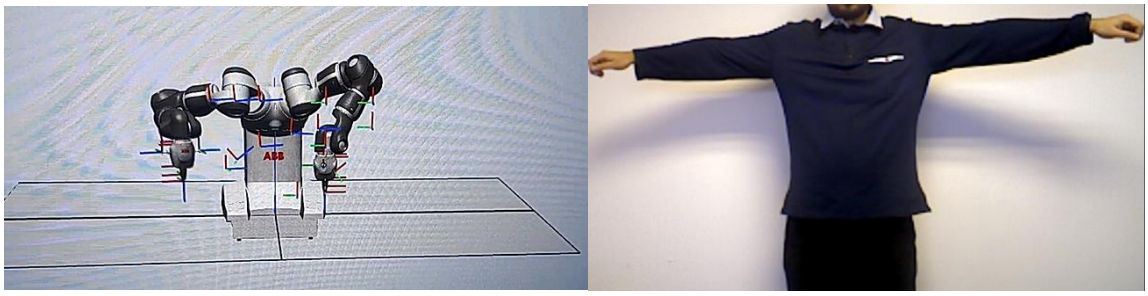


Fig.3.8 Posición del robot vs posición de la persona

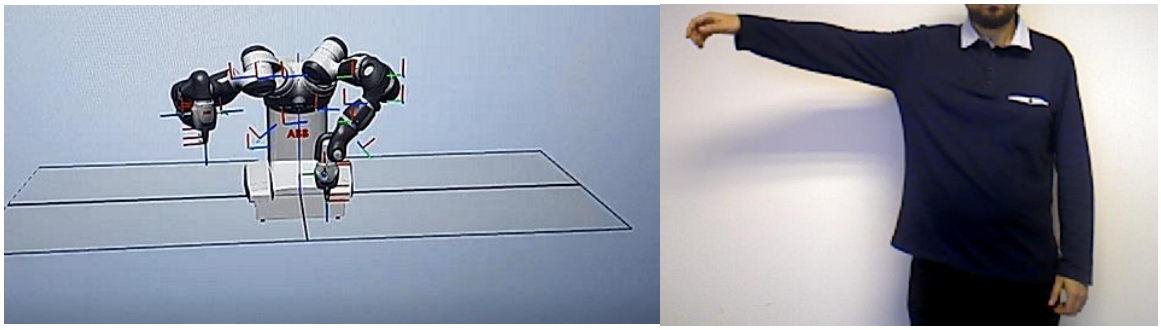


Fig.3.9 Posición del robot vs posición de la persona

3.2. CALIBRACIÓN DE LA CÁMARA

El primer paso para el desarrollo de aplicaciones de visión artificial es la calibración de la cámara. De esta forma se podrá obtener de una manera exacta la posición de los marcadores en el mundo real.

En este caso, la programación de la calibración de la cámara se realiza en el mismo programa que la detección de marcadores. Para diferenciar si se quiere calibrar la cámara o si se quiere detectar los marcadores, a la hora de ejecutar el programa se le pasará unos argumentos diferentes.

A grandes rasgos, en el programa de la calibración de la cámara se realizan los siguientes pasos:

`inputVideo.open()`: abre un dispositivo de captura para la captura de video. Como parámetro de entrada recibe la identificación del archivo que se quiere abrir.

`inputVideo.set ()`: Establece la propiedad en la captura del video. En este caso se establecerá el ancho y la altura de los cuadros en el flujo del video. Los valores del ancho y la altura se pasarán como argumentos al ejecutar el programa.

`cv::cvtColor(view, viewGray, COLOR_BGR2GRAY)`: Convierte la imagen adquirida de un espacio de color a otro. En este caso convierte de BGR (es el formato de color predeterminado en Opencv) a imagen en escala de grises.

`found = findChessboardCorners ()`: Encuentra las posiciones de las esquinas internas del tablero de ajedrez. Como primer parámetro recibe la imagen convertida a escala de grises, el segundo es el número de esquinas interiores por fila y columna del tablero de ajedrez (este parámetro se pasa como argumento en la ejecución del programa), el tercero es la matriz de salida de esquinas detectadas y el cuarto parámetro es una combinación de varios indicadores de operación.

`cornerSubPix ();` Refina las ubicaciones de las equinas.

`drawChessboardCorners ();` Dibuja las esquinas del tablero de ajedrez detectadas. Como parámetros recibe la imagen de destino, el número de esquinas interiores por fila y columna del tablero, la matriz de esquinas detectadas que es la salida de findChessboardCorners y unos parámetros que indica si el tablero completo fue encontrado o no (es el valor de retorno de findChessbordCorners.

Una vez que ya se tienen los puntos de objeto y puntos de imagen ya se procede a la calibración. Para ello se utiliza la siguiente función:

`ret, mtx, dist, rvecs, tvecs= calibrateCamera ()`: Esta función devuelve la matriz de la cámara, los coeficientes de distorsión, vectores de rotación y translación.

Una vez obtenidos los parámetros de la calibración, estos se guardan en un fichero (la dirección de este fichero se pasa como argumento cuando se ejecuta la aplicación.

Para poder realizar la calibración de la cámara se desarrolló un programa usando las funciones comentadas anteriormente al cual se le pasan los argumentos siguientes:

- e=2 → para indicar que se quiere elegir la opción de calibrar la cámara.
- w=9→ número de esquinas interiores de una de las dimensiones del tablero de calibración.
- h=6 → número de esquinas interiores de la otra dimensión del tablero de calibración.
- wC=…→ancho de la resolución de la cámara.
- hC=…→alto de la resolución de la cámara.
- s=0.021→ tamaño de los cuadros del tablero en metros.
- n=30→ número de capturas que se quieren realizar para la calibración.
- d=1500→el retardo en milisegundos entre las diferentes capturas.
- -o=ficheroCalibracionCamara\cameraLogitech640x480.yml→la dirección del fichero donde se guardarán los parámetros de la calibración.

Así pues, se genera un archivo por lotes para que la ejecución de la aplicación sea más fácil y no sea necesario insertar cada vez los argumentos, sino que simplemente se cambien aquellos que se deseen. El contenido del archivo es el siguiente:

```
cd C:\Users\jivae\workspace\OpencvEjemplo\Debug
OpencvEjemplo.exe -e=2 -w=9 -h=6 -wC=640 -hC=480 -s=0.021 -n=30 -d=1500 -o=ficheroCalibracionCamara\cameraLogitech640x480.yml
```

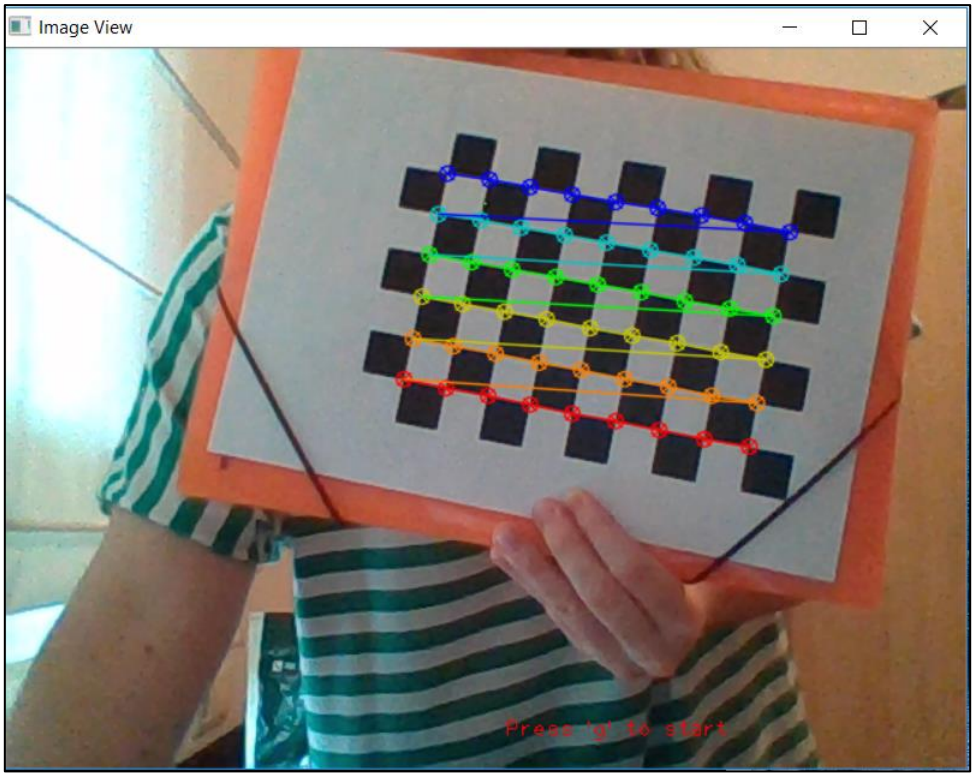


Fig.3.10 Calibración de la cámara

3.3. APLICACIÓN DETECTA-MARCADORES

3.3.1. OBTENCIÓN DE LA POSICIÓN DE LOS MARCADORES

El segundo paso en el desarrollo de esta aplicación es detectar la posición, respecto las coordenadas de la cámara, de los marcadores utilizados.

Para ello lo primero que se debe hacer es elegir los marcadores que se van a utilizar, así como también el tamaño de estos para que puedan ser detectados.

En el caso de la aplicación desarrollada con el robot YuMi, el diccionario que se ha utilizado es el original de ARUco. El marcador número 9 es el que se ha elegido para el brazo izquierdo y el marcador 8 para el brazo derecho.

En el caso de la aplicación desarrollada con el robot IRB 140 el marcador utilizado es el marcador 0 del diccionario 4×4 de ARUco.

Los marcadores se han construido de tal forma que usando un velcro se pueda poner en las muñecas de las personas y de esta forma se puedan manejar fácilmente.

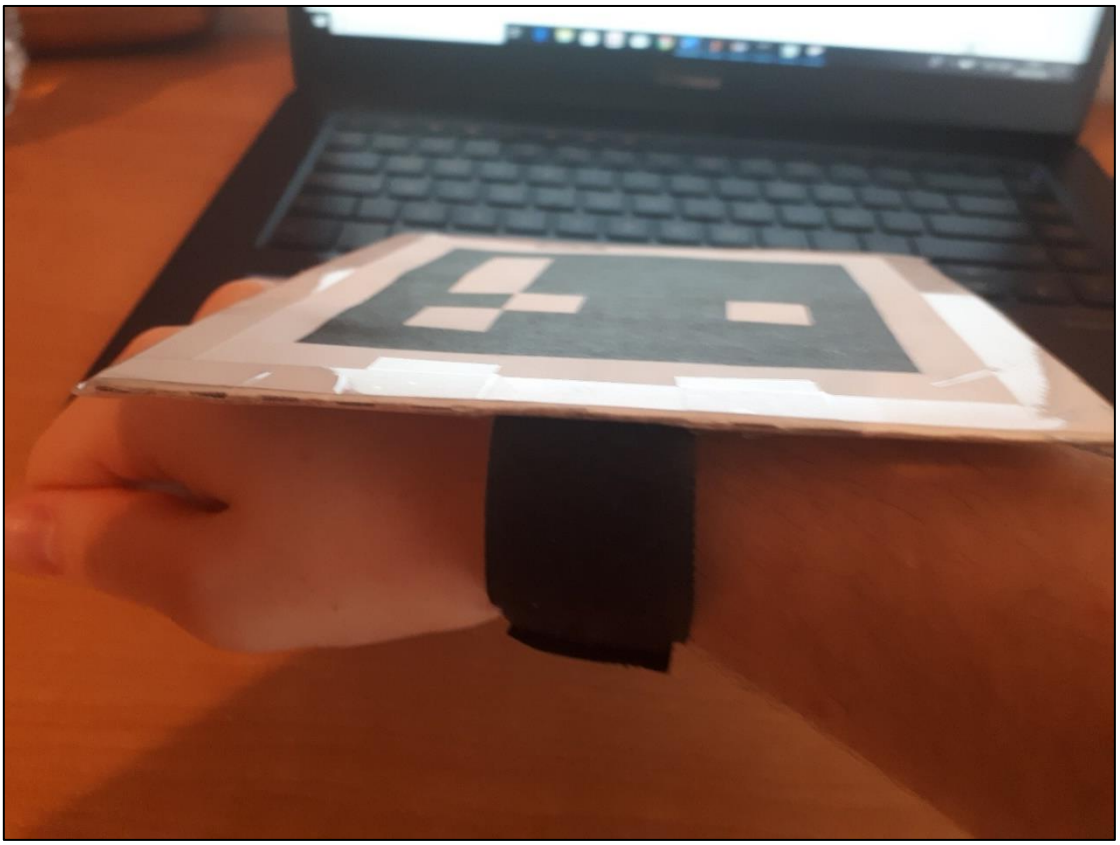


Fig.3.11 Marcador con velcro

Se debe tener en cuenta que durante todo el desarrollo de la aplicación hay casos en los que los marcadores no se detectan. Para estos casos lo que se hace es poner un 1000 en las posiciones correspondientes a ese marcador no detectado. Si se envía un 1000 al robot, este sabrá que no se ha detectado marcador y no se moverá de la posición en la que se encuentra.

Se decidió que la distancia desde la cámara a la persona que mueva sus brazos sea de aproximadamente 1.5m. En primera fase se intentó utilizar unos marcadores de 7cm, pero debido a su reducido tamaño la aplicación de detección de marcadores no era capaz de detectarlos. Por este motivo se pasó a utilizar marcadores de 9cm. Estos marcadores sí que se detectaban a esa distancia, pero la detección no era nada estable ya que al realizar algunos movimientos un poco más rápidos la aplicación no era capaz de reconocer los marcadores durante un periodo corto de tiempo. Esto podía llevar inestabilidad y confusión en la realización de movimientos del robot. Por este motivo, se pasó a utilizar marcadores de 14cm. La aplicación podía detectar estos marcadores a una distancia de 1.5m y la detección también era estable, casi nunca llegaban a perderse.

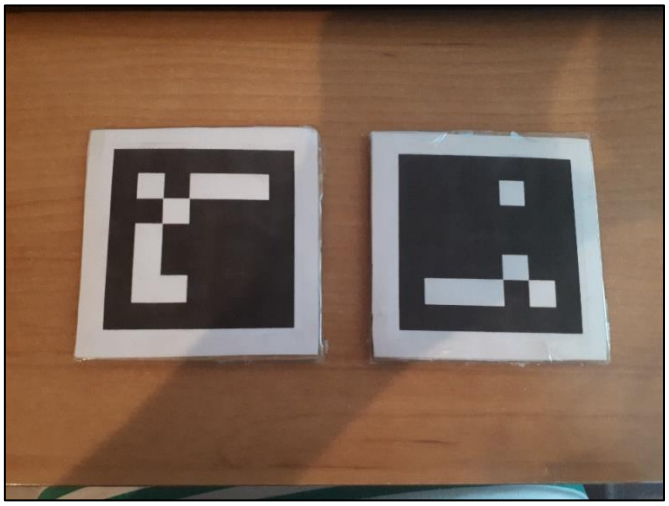


Fig.3.12 Marcadores de 7 cm

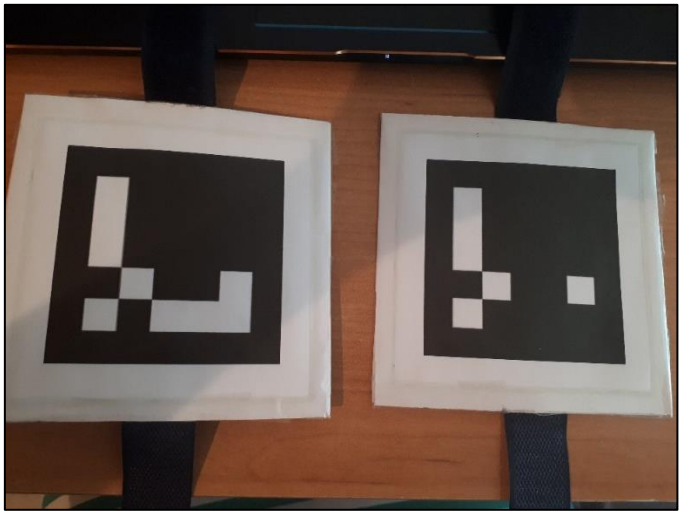


Fig.3.13 Marcadores de 9 cm

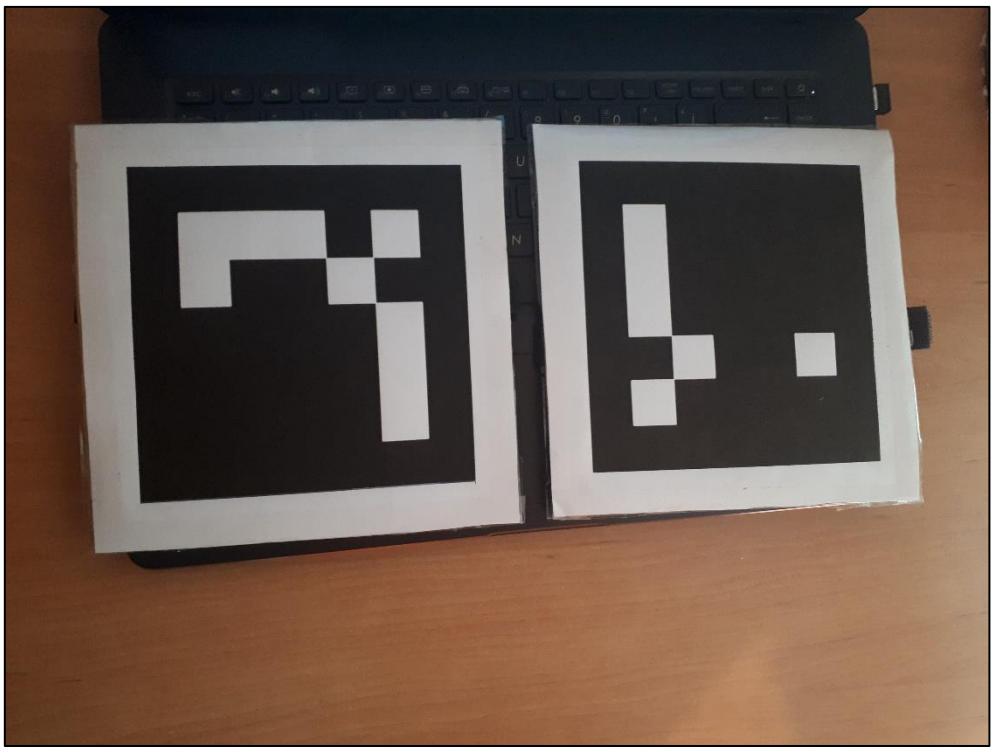


Fig.3.14 Marcadores de 14 cm

El primer paso en el programa realizado es establecer las propiedades en la captura del video:

`inputVideo.set(CAP_PROP_FRAME_WIDTH, ancho);` ancho de los cuadros en el flujo de video (se pasa como argumento en la ejecución de la aplicación)

`inputVideo.set(CAP_PROP_FRAME_HEIGHT, alto);`alto de los cuadros en el flujo de video (se pasa como argumento en la ejecución de la aplicación)

`inputVideo.set(CV_CAP_PROP_AUTO_EXPOSURE, 0.0);` Se deshabilita la propiedad en la que se puede habilitar la auto exposición.

`inputVideo.set(CAP_PROP_GAIN, vGanancia);`ganancia de la imagen (en este caso 160)

`inputVideo.set(CAP_PROP_EXPOSURE, -12);`Exposición (este parámetro es el que se va variando según la luz que recibe la cámara ya que dependiendo de esto se puede o no detectar los marcadores)

`inputVideo.set(CAP_PROP_FPS, fps);`Tasa de cuadros (se pasa como argumento en la ejecución de la aplicación).

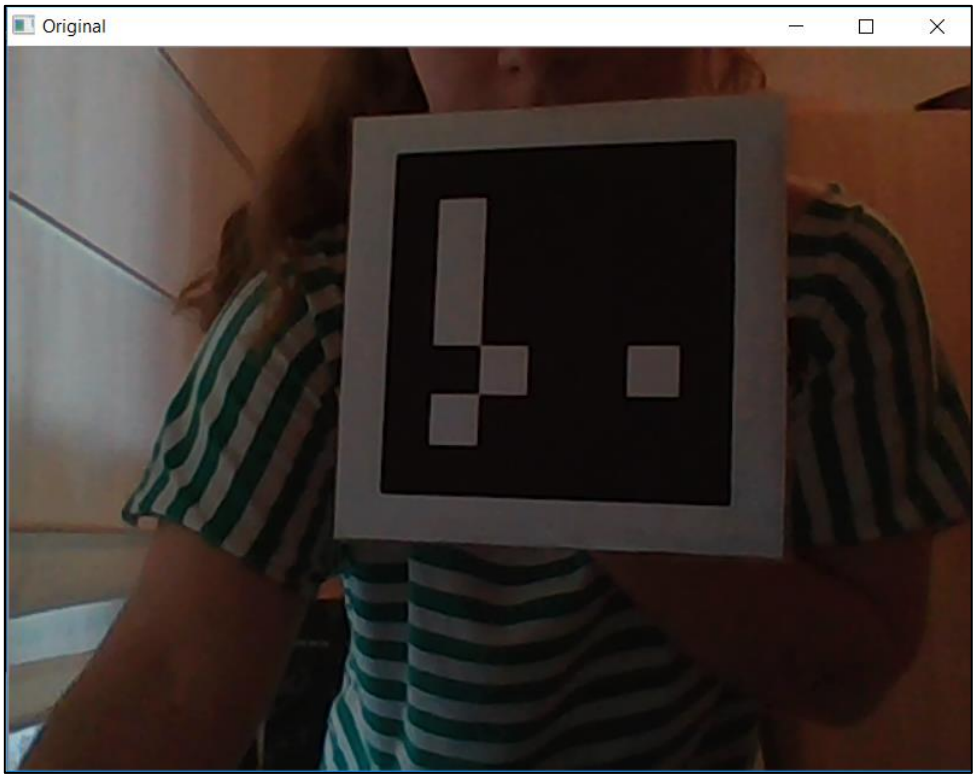


Fig.3.15 Exposure de -5

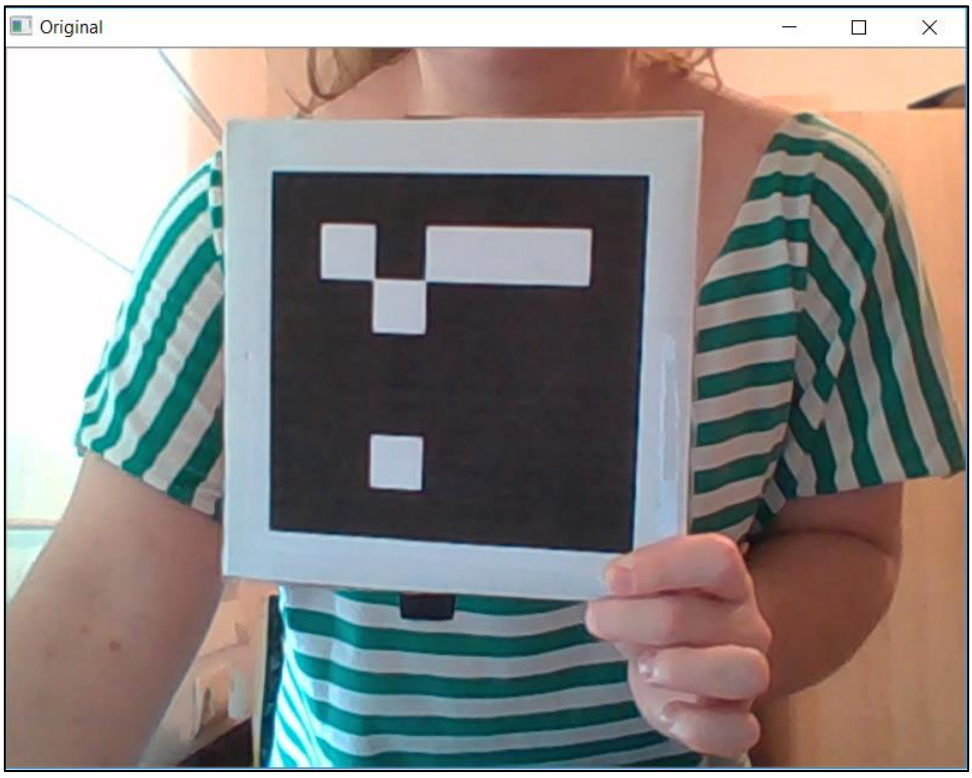


Fig.3.16 Exposure de -2

En el programa se crean dos hilos:

El primer hilo es el encargado de mostrar continuamente por pantalla lo que la cámara captura en el video. Para ello se abre una ventana cuyo título es “Original”. Si mientras esta ventana esta activa se pulsa la tecla ‘v’ se abre una nueva ventana donde se puede ver el video real con los elementos virtuales de los marcadores incluidos. Además, si se pulsa, sobre la ventana “Original”, la tecla ‘s’ se empieza el análisis de los marcadores, es decir, se empieza a obtener la información sobre la posición de los marcadores respecto la cámara.

El segundo hilo es el que se encarga de mostrar por pantalla el resultado de la realidad aumentada (cuando sobre la ventana “Original” se pulsa ‘v’). También es el hilo encargado de obtener la posición de los marcadores y hacer el envío al robot (una vez se haya pulsado la tecla ‘s’ sobre la ventana “Original”).

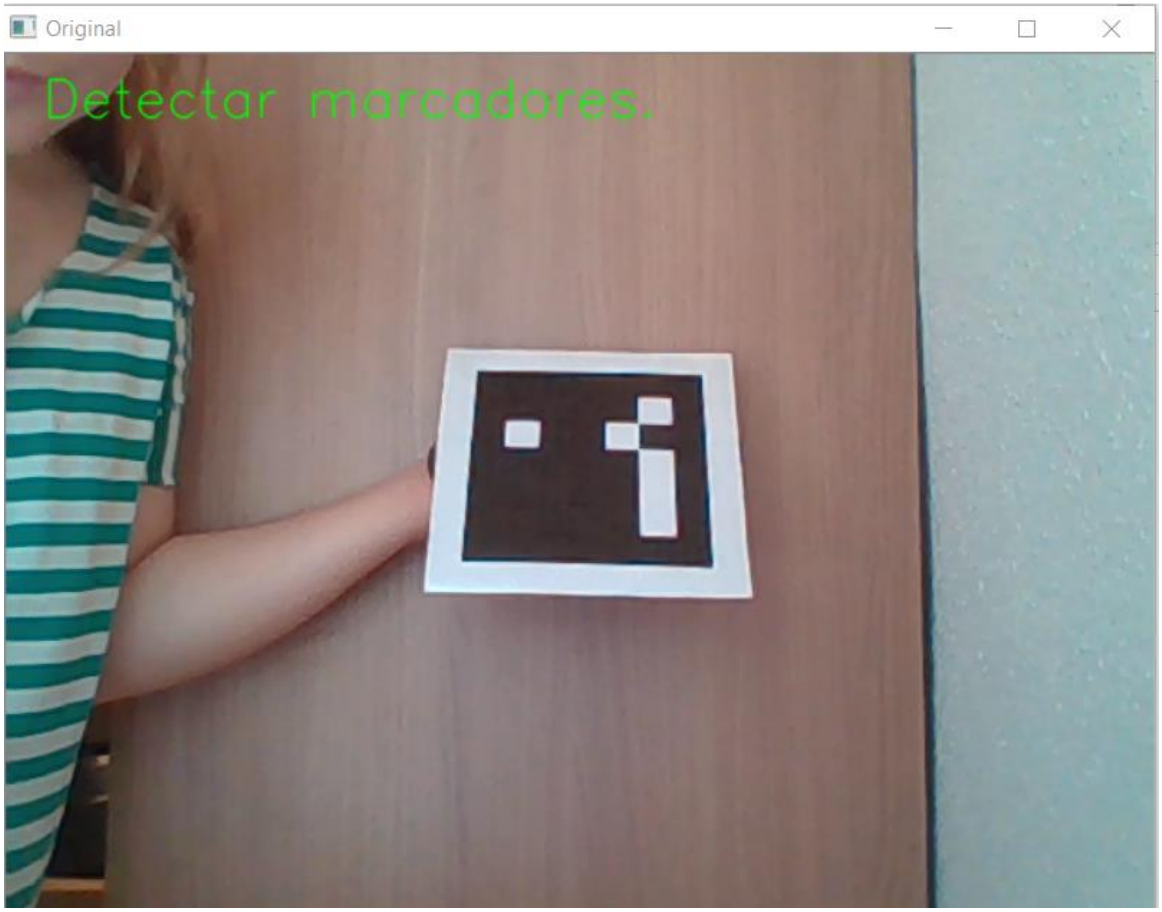


Fig.3.17 Ventana inicial

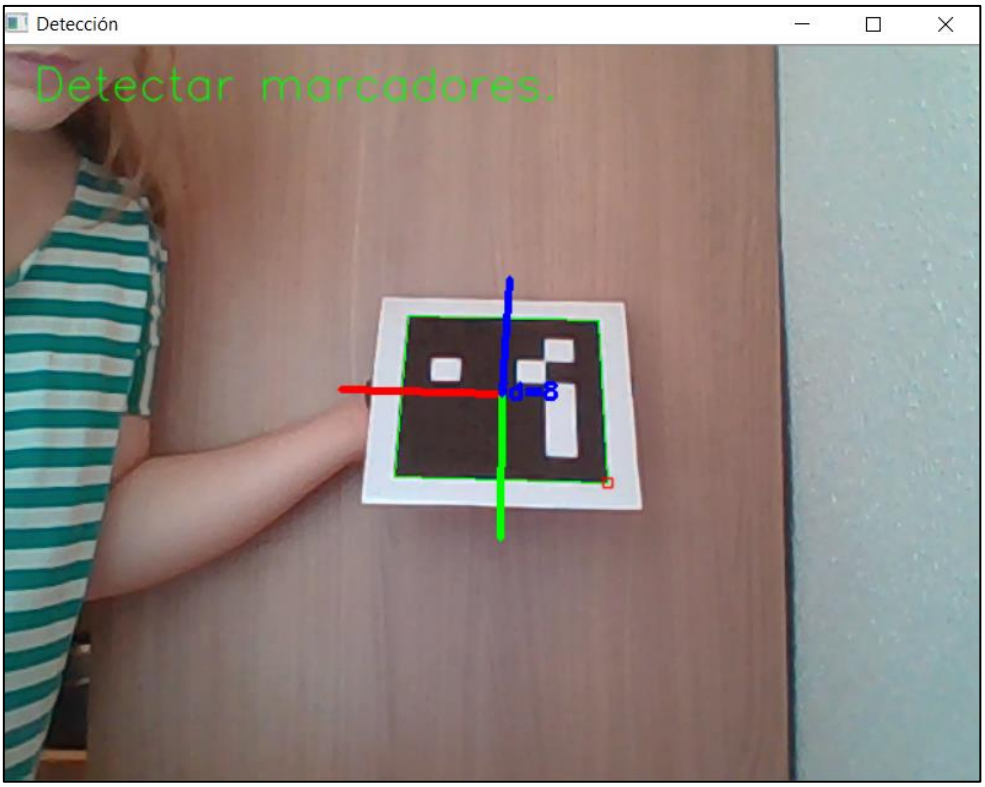


Fig.3.18 Ventana con la realidad aumentada

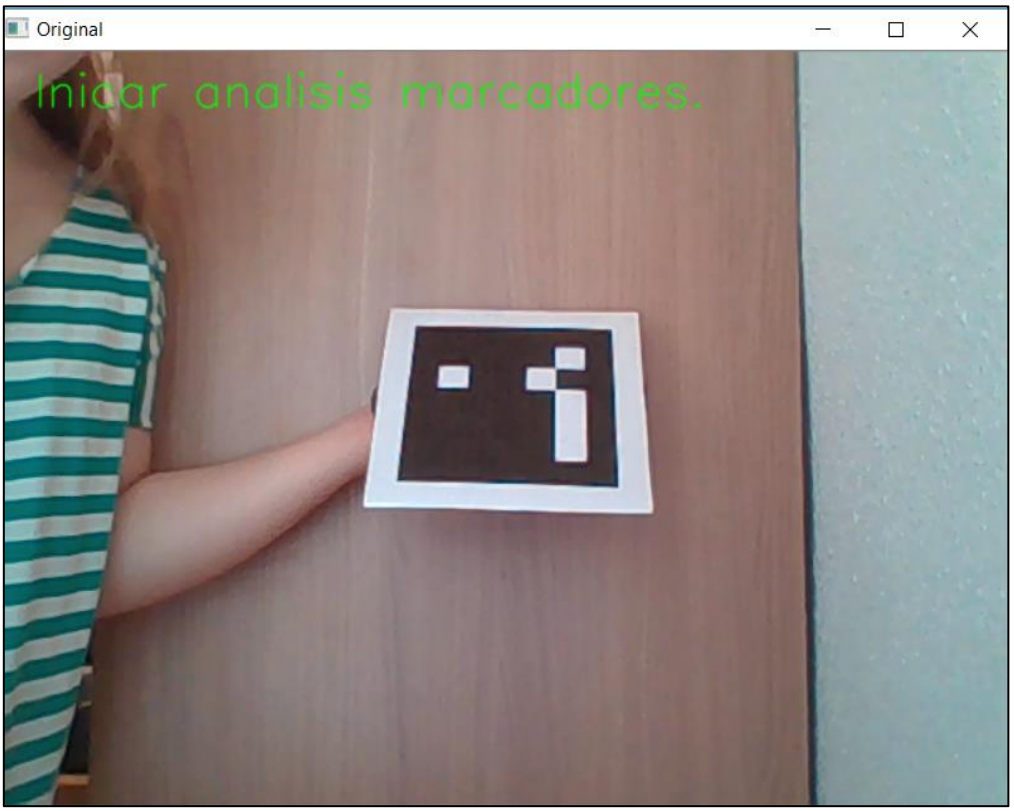


Fig.3.19 Ventana cuando se empieza a enviar

En resumen, en el primer hilo únicamente se muestra en una ventana lo que la cámara captura mientras que en el segundo hilo es donde se hace la detección de los marcadores, la estimación de su posición y la representación de los elementos virtuales sobre los marcadores del mundo real.

A grandes rasgos, la programación relacionada con los marcadores se realiza utilizando las siguientes funciones:

`cv::aruco::detectMarkers()`: Realiza la detección de marcadores en la imagen de entrada. Solo se buscan los marcadores incluidos en el diccionario especificado. Para cada marcador detectado, devuelve la posición 2D de su esquina de imagen y su correspondiente identificador. El primer parámetro que recibe esta función es la imagen de entrada, el segundo es el diccionario de los marcadores a buscar, el tercero es el vector de esquinas marcadoras detectadas, el cuarto es el vector de identificadores de los marcadores detectados y el quinto son los parámetros de detección de marcadores.

`aruco::drawDetectedMarkers(imageCopy, corners, ids)`: Dibuja marcadores detectados en la imagen. Como parámetros de entrada recibe la imagen de entrada, las posiciones de las esquinas del marcador en la imagen de entrada y el vector de identificadores para marcadores en marcadores.

`aruco::estimatePoseSingleMarkers(corners, markerLength, camMatrix, distCoeffs, rvecs, tvecs)`: Estima la posición para los marcadores individuales. El primer parámetros de entrada es el vector de esquinas de marcadores ya detectados, el segundo es la longitud del lado de los marcadores,el tercero es la matriz A obtenida en la calibración, el cuarto es el vector de coeficientes de distorsión, el quinto es la matriz de rotación de salida y el último es la matriz de translación de salida.

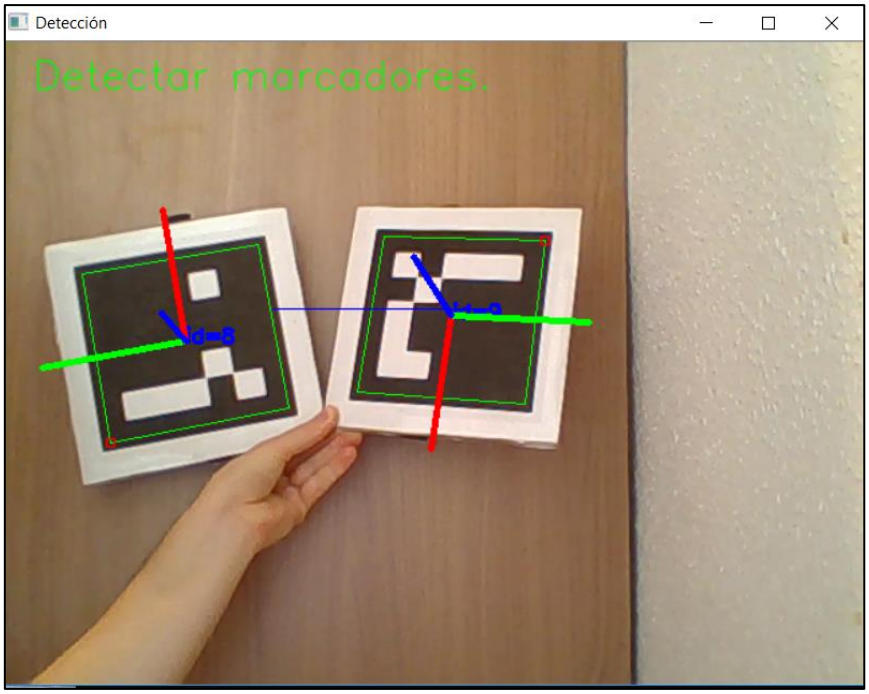


Fig.3.20 Detección de marcadores

Una vez programadas estas funciones, ya podemos ver y obtener la posición de los marcadores que la cámara detecta.

Cuando ya se sabe cómo obtener la posición de los marcadores respecto a la cámara se debe elegir respecto qué sistema de coordenadas se enviarán las posiciones al robot. Como al robot le es imposible llegar al punto (0,0,0) con los dos brazos, enviarle una posición absoluta al robot sería una solución muy poco factible. Por este motivo se ha decidido coger la primera posición de los marcadores como posición inicial y enviar el resto de las posiciones relativas a esta posición inicial. De esta forma lo que se hace es guardar la posición inicial en una variable (en este caso xrobotinicial1, yrobotinicial1, zrobotinicial1).

A continuación, se obtiene la posición del marcador respecto al origen de coordenadas de la cámara (xrobot, yrobot, zrobot) y a esta posición se le resta la posición inicial antes guardada. La información que se le enviaría al robot sería xLsinfiltrar, yLsinfiltrar, zLsinfiltrar.

```
xLsinfiltrar=xrobot-xrobotinicial1;

yLsinfiltrar=yrobot-yrobotinicial1;

zLsinfiltrar=zrobot-zrobotinicial1;
```

Explicado de otra forma, si respecto de la posición inicial el marcador se mueve 50mm en x entonces al robot se le enviaría la información de x=50mm. Así pues, el robot se moverá 50mm respecto su posición inicial.

La posición inicial del robot es una posición que se ha elegido después de hacer varias pruebas y observar que el rango de los movimientos que se pueden hacer, a partir de esa posición, es grande.

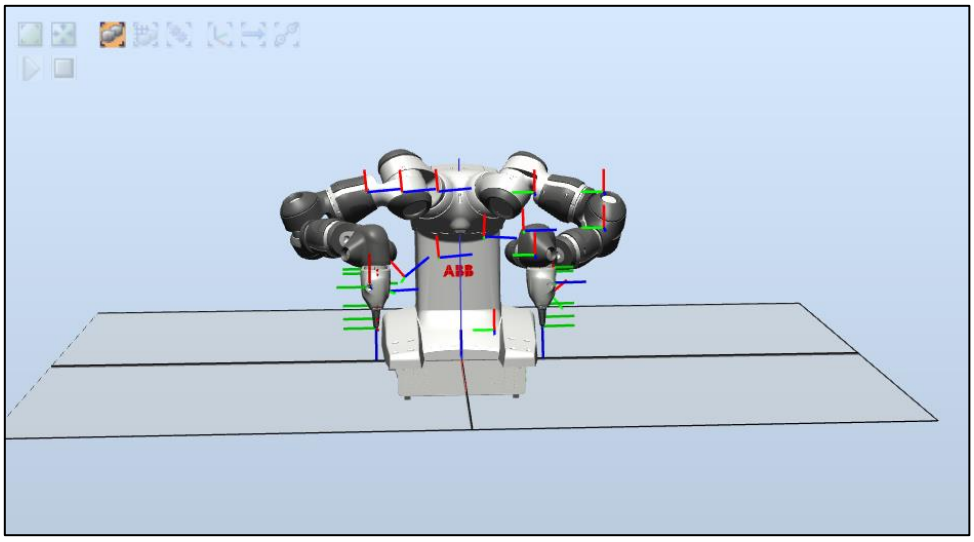


Fig.3.23 Posición inicial del robot YuMi

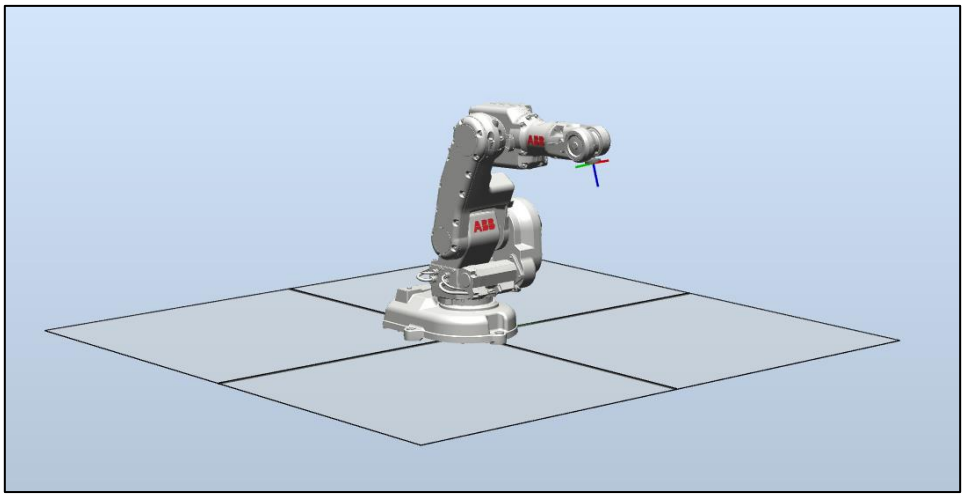


Fig.3.24 Posición inicial del robot IRB 140

En la parte donde se habla sobre las pruebas realizadas con Matlab-RobotStudio (apartado 3.1) se explica como el robot no siempre es capaz de alcanzar las posiciones que los brazos de una persona pueden ya que el área de trabajo del robot es más pequeña que el área de trabajo de los brazos de una persona. Por ejemplo, si la persona extiende del todo su brazo, el robot no pueda llegar a extender su brazo tanto. Por este motivo, las posiciones que se le enviaban al robot estaban divididas por 4 para que el robot llegase a hacer los movimientos. En esta aplicación realizada también se tiene en cuenta esta restricción del robot y se crea una variable, denominada divisor, que indica el número por el cual se divide las posiciones obtenidas de los marcadores. El valor de esta variable se introduce por pantalla una vez lanzada la aplicación. De esta manera si una persona tiene los brazos más grandes el divisor que puede introducir es más grande y de esta forma cuando por ejemplo la persona estire sus brazos el robot también llegará a estirar sus brazos, pero no tanto como los de la persona ya que no es capaz de alcanzar esa posición. El inconveniente que tiene fijar un número grande como divisor, es que para hacer que el robot se mueva, los brazos de la persona se deben desplazar bastante ya que al robot le llega el valor del desplazamiento de la persona dividido por un número. Por tanto, le llega el valor de desplazamiento más pequeño que el de la persona. Por este motivo, en las pruebas realizadas se ha fijado el número 1 como divisor, enviándole así al robot el mismo valor del desplazamiento que la persona con los marcadores ha realizado. Como las personas que han probado las aplicaciones han tenido un tamaño de brazos normal, poner 1 como divisor no ha dado problemas. En el caso de que la persona que lo pruebe tenga los brazos más grandes el valor del divisor deberá subir.

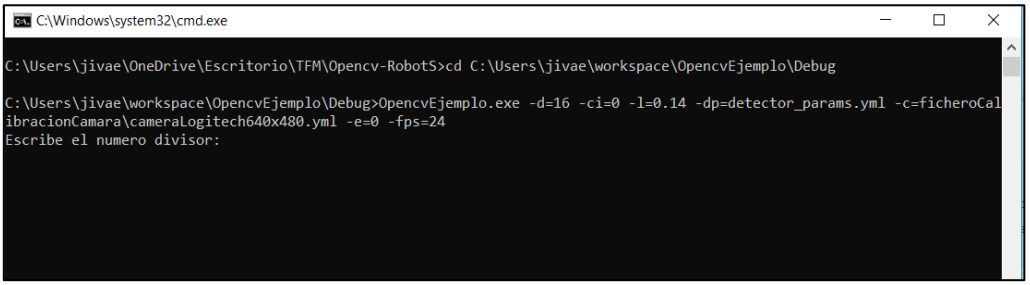


Fig.3.25 Introducción del divisor

Otra cosa que se debe tener en cuenta es que cuando se obtienen las posiciones de los marcadores, el valor está en metros, pero al robot se le deben enviar posiciones en milímetros por tanto antes de enviar valores al robot, las posiciones obtenidas de los marcadores se deben multiplicar por 1000 para pasarlas a milímetros.

La función de ARUco obtiene la posición de los marcadores respecto la cámara, pero se necesita tener los valores respecto las coordenadas del robot. También se ha pensado que la forma más intuitiva de mover el robot es hacerlo en espejo, es decir que cuando se desplace el brazo derecho de la persona el robot mueva su brazo izquierdo y cuando se desplace el izquierdo de la persona el robot mueva su brazo derecho. Así pues, sabiendo los ejes de coordenadas de la cámara y los del robot se obtiene la siguiente relación:

$X_{robot}=Z_{cámara}$

$Y_{robot}=X_{cámara}$

$Z_{robot}=Y_{cámara}$

La programación para obtener la posición respecto las coordenadas del robot teniendo en cuenta que los movimientos se hacen en espejo tiene la siguiente expresión:

```
double xrobot=(1000* encontrado->second.val[2])/divisor;
double yrobot=(-1000* encontrado->second.val[0])/divisor;
double zrobot=(-1000*encontrado->second.val[1])/divisor;
```

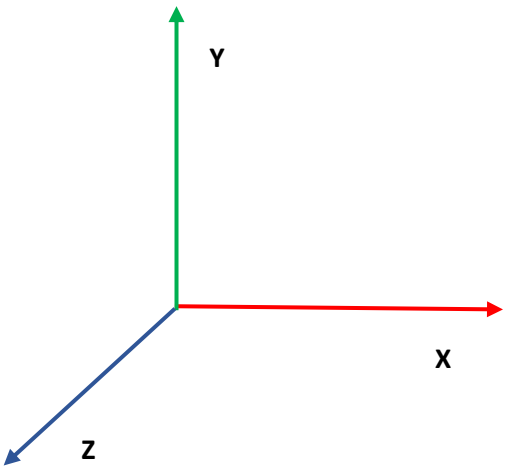


Fig.3.26 Ejes de la cámara

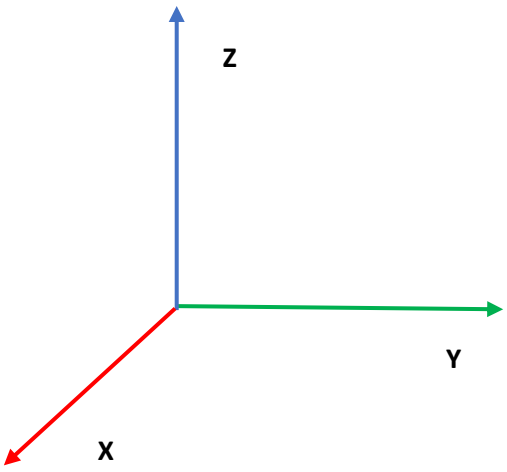


Fig.3.27 Ejes del robot

3.3.2. Envío de la posición de los marcadores

Tal y como se ha comentado anteriormente, la aplicación detecta-marcadores funcionará como cliente dentro de la comunicación que se realiza entre ésta y el robot.

Los pasos que se han seguido para establecer la comunicación son los siguientes:

1. Se abre un socket utilizando la función *socket ()*. Esta función retorna un descriptor de socket, que es de tipo *int.* Si existe algún error en la creación, retorna -1 y la variable global *errno* se establece con un valor que indica el error que se produjo.

   ```
   conexion = socket (int dominio, int tipo, int protocolo);
   ```

   conexión: es el descriptor de socket que devuelve la función. Este valor se utilizará posteriormente para conectarse, recibir conexiones, enviar y recibir datos.

   dominio: dominio en el que se realiza la conexión. En este caso será AF_INET ya que el servidor y el cliente se encuentran en máquinas diferentes.

   tipo: puede ser SOCK_STREAM, SOCK_DGRAM SOCK_RAW o SOCK_SEQPACKET, dependiendo del tipo de socket que se quiera crear. En este caso es SOCK_STREAM porque nos proporciona una comunicación fiable.

   protocolo: al poner esta variable a 0 el sistema selecciona automáticamente el protocolo más apropiado.

   Así pues, en este caso la función se ha declarado de la siguiente forma:

   ```
   conn_socket=socket (AF_INET,SOCK_STREAM, 0);
   ```

2. Después de abrir el socket se declaran estructuras tipo *sockaddr* que contienen la información sobre la IP y el puerto del servidor.

   ```
   server.sin_family = AF_INET;

   server.sin_port = htons(7654);

   server.sin_addr.s_addr=inet_addr("192.168.2.103");
   ```

3. Una vez abierto el socket, se solicita la conexión con el servidor utilizando la función *connect ().* Dicha función queda bloqueada hasta que el servidor acepte la conexión. En el caso de que no haya servidor, se sale dando un error. Tiene la siguiente estructura:

   ```
   connect(int conexion,struct sockaddr *serv_addr,int addrlen)
   ```

   conexión: es el descriptor del socket devuelto por la función socket ()

   serv_addr: es una estructura sockaddr que contiene la dirección IP y número de puerto del servidor.

addrlen: tamaño de la estructura sockaddr. Se utiliza sizeof(struct sockaddr)

La llamada de esta función, en el proyecto, se realiza de la siguiente forma:

```
connect(conn_socket,(struct sockaddr*)&server,sizeof(server)
```

Una vez establecida la conexión, se puede empezar la transferencia de datos. *Send( )* y *recv ( )* son las dos funciones que sirven para la transferencia de datos en los sockets STREAM.

Tienen la siguiente estructura:

Para enviar datos se utiliza:

```
send(int conexion, const void*msg,int len, unsigned int flags)
```

conexión : descriptor del socket por el cual se enviarán los datos.

msg : puntero a los datos que se quieren enviar.

len : longitud de los datos en bytes.

flags : se pone por defecto el valor de 0.

La función retorna la cantidad de datos que se han enviado, que puede ser menor que la cantidad de datos que se escribieron en el buffer para enviar. Enviará la cantidad máxima de datos que pueda manejar. El programador tiene que asegurarse del correcto envío de datos, comparando la cantidad de datos enviado con *len*. Si no se han enviado todos los datos, se pueden enviar en la próxima llamada a esta función.

Para recibir datos se utiliza:

```
recv(int conexion, void *buf,int len, unsigned int flags)
```

conexión : descriptor del socket por el cual se recibirán los datos.

buf : puntero a un buffer donde se almacenarán los datos recibidos.

len : longitud del buffer *buf.*

flags :se pone por defecto el valor 0.

Si no existen datos para recibir en ese socket, la llamada de la función *recv ( )* se bloquea (no retorna nada) hasta que llegan los datos. Existe la opción de hacer que esta función no sea bloqueante de forma que cuando no hay datos para recibir, la función retorna un -1 y establece la variable *errno*= EWOULDBLOBK.
Retorna el número de bytes recibidos.

3.4. PROGRAMACIÓN DEL ROBOT IRB 140

Como ya se ha explicado en el objetivo del proyecto, se pretende desarrollar una aplicación para la tele operación de robots. En este caso se han probado dos robots, el IRB 140 que presenta un solo brazo manipulador y el YuMi que presenta dos brazos manipuladores. En este apartado se va a explicar la programación del IRB 140.

A diferencia de la programación del YuMi, el IRB 140 al presentar un solo brazo puede programarse utilizando una sola tarea.

En este caso lo primero que se hace en la tarea es mover el robot al punto inicial y abrir el socket:

```
MoveJ punto_inicial,v50,z1,tool0\WObj:=wobj0;

SocketCreate socket_server;

SocketBind socket_server, server_ip, puerto;

SocketListen socket_server;

SocketAccept socket_server, socket_client\ClientAddress:=client_ip;
```

A continuación, se entra en un bucle en el cual se hace lo siguiente:

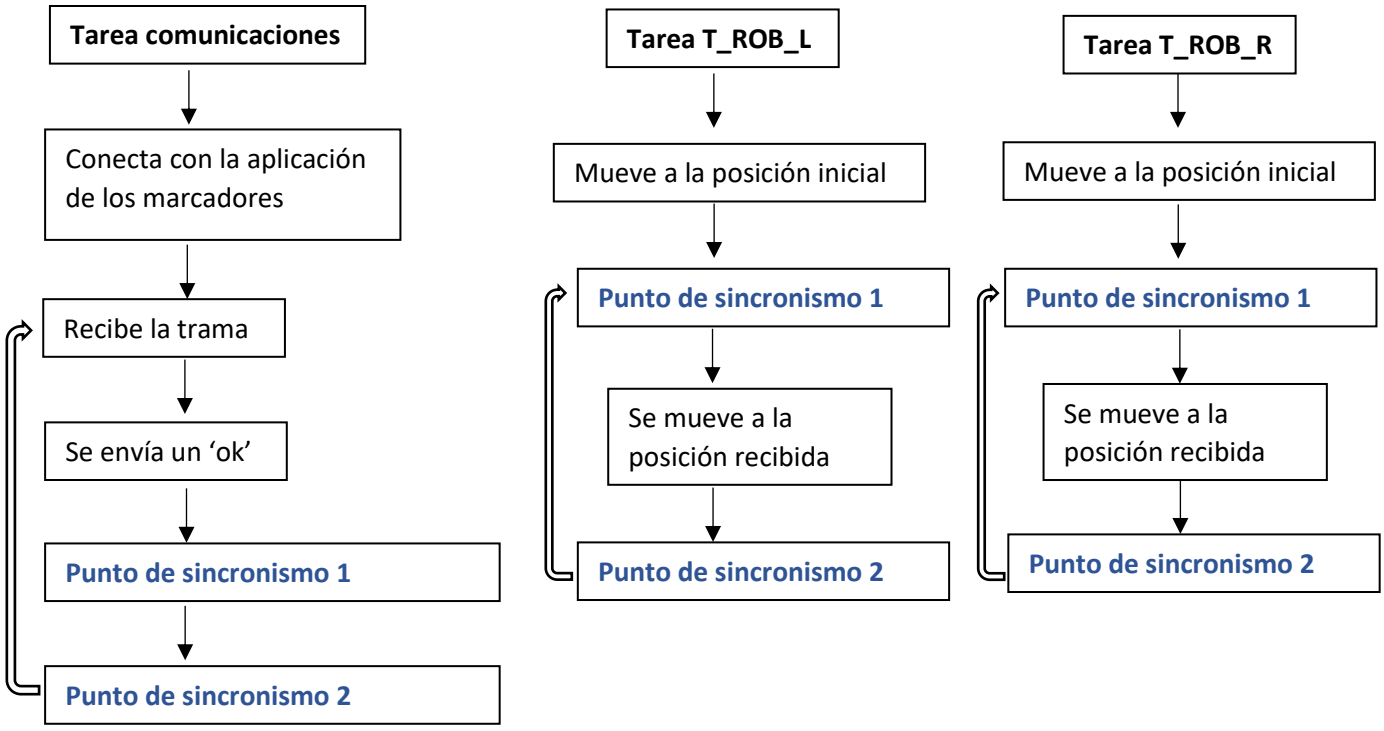
- Se reciben los valores enviados por la aplicación detecta-marcadores y se asignan a diferentes variables según sea la posición x y o z.
- Se envía la aplicación detecta-marcadores un ‘ok’ para indicarle que ya puede enviar otra trama
- Se mueve el brazo a los valores de las posiciones recibidas.

```
lectura_socket;

SocketSend socket_client\Str:=cadena_enviar;

IF coorXL=1000 THEN

ELSE

MoveL Offs(puntoinicial,coorXL,coorYL,coorZL),v50,z10,

tool0\WObj:=wobj0;

ENDIF
```

3.5. PROGRAMACIÓN DEL ROBOT YUMI

El lenguaje de programación del YuMi es el RAPID. A diferencia de los otros robots que tienen un solo brazo, este robot debe mover los dos brazos a la vez. Por este motivo se han creado dos tareas, cada una de ellas mueve un brazo diferente. Además, debe existir otra tarea que sea la encargada de comunicar con la aplicación que detecta los marcadores. Cuando se crea más de una tarea en la programación del robot, éstas se ejecutan concurrentemente.

La idea con la que se ha partido para la programación de las tres tareas es la siguiente:



Tal y como se puede observar en el esquema existen diversos puntos de sincronismo. Estos puntos son puntos donde las tareas especificadas en `YuMi_App_task_list`, detienen su ejecución hasta que todas las tareas de la lista lleguen a este punto. Así por ejemplo en la tarea de comunicaciones una vez se envía el 'ok', se entra en el punto de sincronismo 1. El programa dentro de esta tarea se queda parado hasta que las otras dos tareas acabaran de moverse a la posición inicial y se encuentren dentro del punto de sincronismo 1. Una vez todas las tareas hayan llegado al punto de sincronismo 1 se ponen nuevamente en marcha las tres tareas. De la misma forma ocurre con el punto de sincronismo 2. Si por ejemplo el brazo derecho ya se ha movido a la posición deseada y se encuentra en el punto de sincronismo 2, la tarea espera a que comunicaciones llegue al punto de sincronismo 2 y la tarea del brazo izquierdo también.

Esto se hace para que todas las tareas se coordinen y una vez se haya recibido la trama con las posiciones deseadas de los dos brazos, estos puedan moverse a la posición y en el caso de que alguno de los brazos haya terminado sus movimientos antes, se espere a que el otro brazo también termine

sus movimientos y entonces ya se pueda recibir una nueva trama. De esta forma se consigue asegurar que cada brazo hace simultáneamente los movimientos recibidos en la trama que envía la aplicación de los marcadores.

Para crear los puntos de sincronismo lo primero que se debe hacer es crear una lista con todas las tareas que intervendrán en el sincronismo. En este caso serán las tres tareas creadas en el proyecto.

```
PERS tasks YuMi_App_task_list_B{3}:=[["comunicaciones"], ["T_ROB_L"],
["T_ROB_R"] ];
```

A continuación, se crearán las variables de los diferentes puntos de sincronismo, en este caso será el sincronismo 1 y el sincronismo 5:

```
VAR syncident YuMi_App_sync1_B;

VAR syncident YuMi_App_sync5_B;
```

Este procedimiento se debe realizar en cada una de las tres tareas del proyecto.

3.5.1. Tarea “comunicaciones”

La tarea de comunicaciones es la encargada de recibir la información que envía la aplicación que detecta la posición de los marcadores.

En esta tarea lo primero que se hace es abrir un socket para conectarse con la aplicación. En este caso el socket del robot será el servidor.

```
SocketCreate socket_server;

SocketBind socket_server, server_ip, puerto;

SocketListen socket_server;

SocketAccept socket_server, socket_client\ClientAddress:=client_ip;
```

Una vez hecha la conexión, se entra dentro de un bucle que sigue los siguientes pasos:

- Se lee la trama recibida y se les asignan valores a las variables compartidas con las demás tareas. En el caso de la tarea T_ROB_L y COMUNICACIONES comparten las variables correspondientes a la posición (x y z) del brazo izquierdo. Mientras que la tarea T_ROB_R y COMUNICACIONES comparten las variables que se corresponden a la posición (x y z) del brazo derecho. Para la lectura de la trama se han realizado varias versiones con el fin de aumentar la rapidez en la comunicación y en el movimiento del robot. Estas versiones se explicarán en los apartados siguientes.

- A continuación, se envía un string ‘ok’ para indicarle al programa de los marcadores que ya se ha recibido la trama y que puede volver a enviar otra.

- Se sitúa el punto de sincronismo 1 para esperar a que las demás tareas también lleguen a ese punto.
  ```
  WaitSyncTask YuMi_App_sync1_B, YuMi_App_task_list_B;
  ```

- Finalmente se pone el punto de sincronismo 2 para esperar a que las otras tareas también lleguen a ese punto después de haber realizado los movimientos deseados.

  ```
  WaitSyncTask YuMi_App_sync5_B, YuMi_App_task_list_B;
  ```

Una vez llegado a este punto el bucle se vuelve a repetir.

3.5.1.1. Recepción de tramas. Primera versión.

En los apartados anteriormente explicados, en el caso del robot YuMi, se llegó a la conclusión que lo mejor era enviar la información de cada brazo en una trama diferente [Fig. 3.1].

Por tanto, la lectura del socket se realizaba dos veces seguidas, en la primera recibía los valores del brazo izquierdo y en la segunda los valores del brazo derecho. Para saber de qué brazo se trataba y asegurarse de que es el correcto se verificaba si el primer valor recibido es el 1 (brazo izquierdo) o el 0 (brazo derecho). Después de verificar de que brazo eran los valores, estos se asignaban a las variables compartidas correspondientes.

Esta versión producía un poco de retraso en cuanto a la recepción y a los movimientos del robot ya que debía esperar hasta recibir dos tramas para poder moverse. Así que se pensó en realizar una segunda versión.

3.5.1.2. Recepción de tramas. Segunda versión

La nueva trama que se ha pensado enviar tiene el siguiente aspecto:

xxxx;yyyy;zzzz;xxxx;yyyy;zzzz;

xxxx;yyyy;zzzz;xxxx;yyyy;zzzz;

.

.

.

En una única trama se envía primero las posiciones (x y z) del brazo izquierdo y seguidamente las posiciones (x y z) del brazo derecho. De esta forma no habrá que esperar a recibir dos tramas diferentes para que se pueda mover el robot.

Para hacer la lectura y asignarle a cada variable compartida su valor correspondiente se van leyendo valores hasta llegar al ';'. Así por ejemplo se leen valores hasta llegar al primer punto y coma. Cuando ya se ha llegado se le asigna a la variable de posición x del brazo izquierdo, el valor que se ha leído hasta el punto y coma. Seguidamente se leen valores hasta el otro punto y coma y se el valor leído se

le asigna a la variable de posición y del brazo izquierdo. Este procedimiento se realiza hasta llegar al último punto y coma.

3.5.1.3. Recepción de tramas. Tercera versión

Como ya se ha explicado en el apartado anterior, para asignar valores a las posiciones a las que se debe mover el robot se van leyendo los valores de la trama que se recibe. Pero esto también consume un poco de tiempo ya que se necesita ir recorriendo posición por posición hasta llegar al punto y coma y después asignar valores. Por este motivo se ha decidido enviar y recibir tramas que tengan una estructura fija y de esta forma saber que por ejemplo los primeros cuatros dígitos corresponden a la posición x de la izquierda, los siguientes cuatro a la posición de la izquierda y así sucesivamente. De esta manera la nueva trama se construirá de la siguiente forma, teniendo cada una de las posiciones cuatro dígitos. Aunque la posición solo tenga un dígito la trama se llenará con ceros a la izquierda.

[±xxxx; ±yyyy; ±zzzz; ±xxxx; ±yyyy; ±zzzz;]

[±xxxx; ±yyyy; ±zzzz; ±xxxx; ±yyyy; ±zzzz;]

.

.

.

3.5.2. Tarea "T_ROB_L"

Esta tarea corresponde a los movimientos del brazo izquierdo del robot. El primer paso es llevar a la posición inicial el brazo. Una vez se sitúe en este punto se pasa al punto de sincronismo 1, este punto espera o se asegura que el brazo derecho también ha terminado de llegar a la posición inicial. Una vez las tres tareas hayan llegado a este punto de sincronismo se pasa a realizar el movimiento de desplazamiento a la posición que le ha llegado de la aplicación de los marcadores. Como la información que le llega al robot es el valor del desplazamiento de los brazos respecto el punto inicial, para moverlo también se utiliza el comando que mueve sus brazos a la posición deseada, pero respecto al punto inicial del brazo del robot.

```
MoveL Offs(pos_iniL_agarre,coorXL,coorYL,coorZL),v7000,z10,

Camera\WObj:=wobj0;
```

Cuando haya llegado a la posición recibida se pasa al punto de sincronismo 2. Este punto espera o se asegura que el otro brazo también ha llegado al punto deseado y de esta forma se vuelve a iniciar el bucle. De esta forma se asegura que los dos brazos se mueven sincronizadamente sin perderse ningún movimiento.

En el caso de que la posición recibida sea 1000, eso significa que no se ha detectado el marcador y por tanto el brazo se queda parado hasta volver a recibir otro valor diferente de 1000.

3.5.3. Tarea “T_ROB_R”

La tarea correspondiente a los movimientos del brazo derecho sigue la misma estructura que la correspondiente al brazo izquierdo.

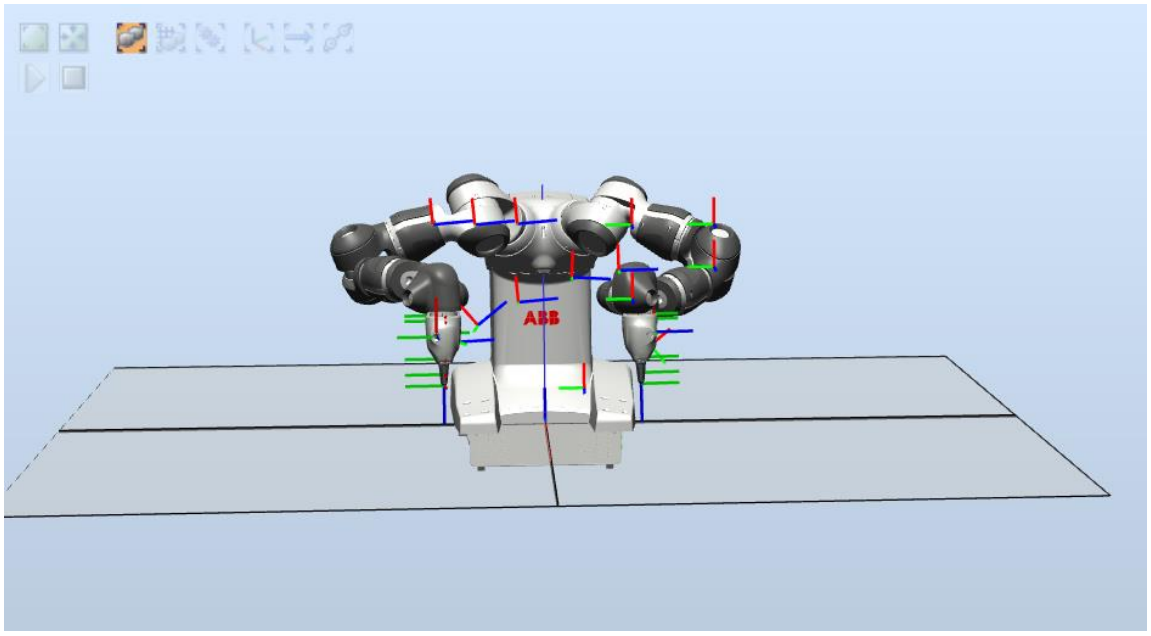


Fig.3.28 Posición inicial del robot

3.5.4. ÁREA DE TRABAJO DEL ROBOT

En muchas ocasiones los brazos de las personas pueden alcanzar mucho más que los brazos del robot, en este caso el robot recibe la orden de desplazarse a una posición que no consigue alcanzar y se parará la ejecución del programa debido a este error. Para evitar esto se decide definir un área de trabajo del robot.

En el caso de que el robot reciba una posición que esté fuera de su área de trabajo, este no hará caso a la posición recibida, sino que simplemente se moverá hasta el máximo que puede alcanzar dentro de esta área. De esta forma se evitará que el robot se pare dando errores de movimientos.

Moviendo manualmente el robot se ha podido observar qué coordenadas puede alcanzar como máximo. A estas coordenadas se les ha restado la posición inicial del robot y los resultados se han definido como valores máximos que puede recibir el robot de la aplicación de los marcadores. En el caso de que se reciban valores superiores a los antes definidos el robot no se moverá a estas posiciones, sino que únicamente se desplazará hasta los máximos definidos.

Los valores del área de trabajo son los siguientes:

```
VAR num maxXL:=80;

VAR num maxYL:=160;

VAR num maxZL:=320;

VAR num minXL:=-170;

VAR num minYL:=-170;

VAR num minZL:=-100;

VAR num maxXR:=110;

VAR num maxYR:=-180;

VAR num maxZR:=320;

VAR num minXR:=-180;

VAR num minYR:=219;

VAR num minZR:=-100;
```

3.5.5. ABERTURA Y CIERRE DE LAS PINZAS

Una vez se consigue mover el robot imitando los brazos de la persona que está en frente de la cámara, abrir y cerrar las pinzas del robot podría suponer una mejora en la aplicación. De esta forma si se llega a la posición deseada se podrán abrir y cerrar las pinzas con el fin de poder coger cosas y trasladarlas de un punto a otro. Por este motivo se ha decidido implementar un control de las pinzas también.

Para realizar este control se han conectado dos pedales a las entradas digitales del robot. De esta forma cuando alguno de los pedales se pulse, el robot recibirá un 1 de esa señal. Mientras no esté pulsado el pedal, la señal estará a 0.

En el YuMi las pinzas son “inteligentes” por tanto solo bastará con indicarle que cierre las pinzas y este lo hará hasta detectar que ya ha conseguido coger las cosas. Pero una de las desventajas de la programación de las pinzas es el hecho de que cada vez que se entra en la orden de abrir o cerrar, estas funciones consumen tiempo y producen un retraso en el movimiento del robot. Para evitar que continuamente se esté perdiendo tiempo al leer la señal digital y decidir si se abre o se cierra la pinza, lo que se hace es lo siguiente:

Si se detecta que no se ha pulsado el pedal y la pinza ya está cerrada, ya no se ejecuta la orden de cerrar pinzas. En cambio, si se detecta que no se ha pulsado y la pinza está abierta, se ejecuta la orden de cerrar pinza.

En el caso de que se haya pulsado el pedal y la pinza no está abierta, se realiza la orden de abrir pinza. Mientras que en el caso de que se pulse el pedal y la pinza ya está abierta la función de abrir pinzas ya no se ejecuta.

```
IF custom_DI_0=1 AND Rcerrada=1  THEN
         g_GripOut;
         Rcerrada:=0;
ENDIF

IF custom_DI_0=0 AND Rcerrada=0 THEN
         g_GripIn;
         Rcerrada:=1;
ENDIF
```

Fig.3.29 Robot YuMi con pinzas

Fig.3.30 Pedal conectado al robot

3.6. DISEÑO DEL FILTRO BUTTERWORTH

Probando el funcionamiento de la aplicación, se ha podido observar que cuando los brazos están parados, el robot realiza ligeros movimientos oscilatorios y no se queda parado, por lo tanto,no presenta mucha precisión. El motivo puede ser debido a que los brazos de una persona no están perfectamente parados ya que por el simple hecho de respirar estos tienden a moverse un poco. Esto produce que la detección de la posición de los marcadores tampoco sea estable ya que detecta que se ha movido unos milímetros. Así pues, el robot recibe pequeñas variaciones de las posiciones y por este motivo se produce un temblor que le quita precisión.

Para solucionar este problema se ha decidido implementar un filtro de Butterworth para las posiciones que se envían al robot.

En primer lugar, se mide el tiempo que se tarda entre un envío y otro de las posiciones de los marcadores. Esto se hace para sacar el periodo de muestreo que en este caso es de 0.05 segundos. Equivale a una frecuencia de 20 Hz.

Una vez obtenida la frecuencia de muestreo se guardan en un fichero datos de las posiciones de los marcadores para analizarlos posteriormente y diseñar el filtro. En este caso se trabaja sobre los datos obtenidos de la posición x del brazo izquierdo.

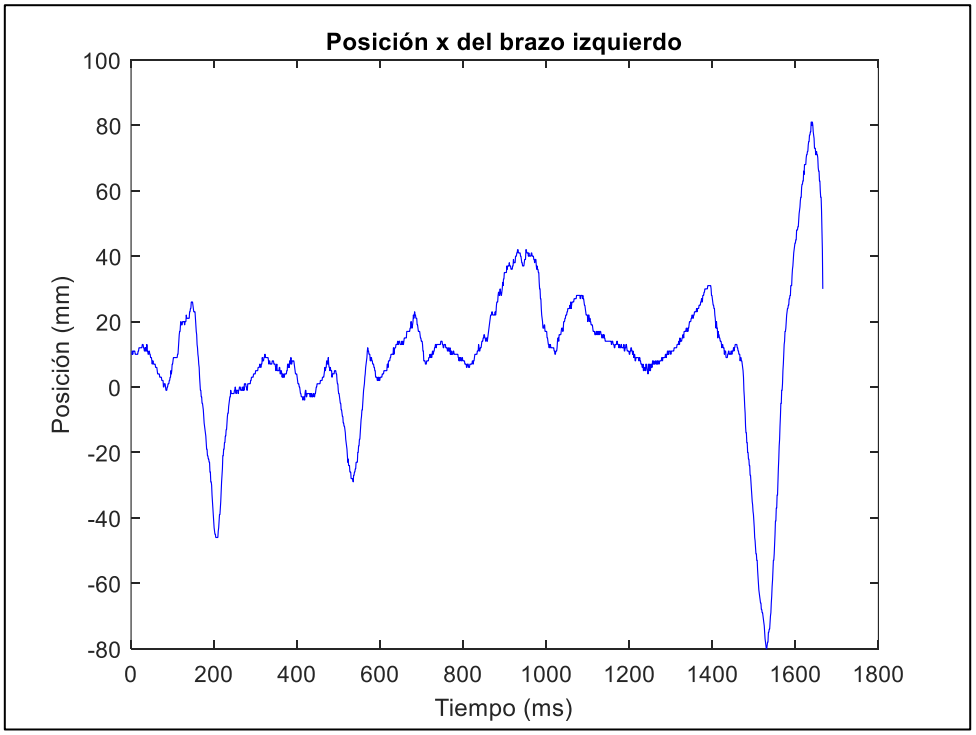


Fig.3.31 Posición x del brazo izquierdo

Como se puede observar en la representación de las posiciones en un periodo corto de tiempo hay pequeñas variaciones que provocan que el robot se mueva.

Para hacerse una idea de cuál podría ser la frecuencia de corte para el filtro, se dibuja la transformada de Fourier de la señal obtenida anteriormente:

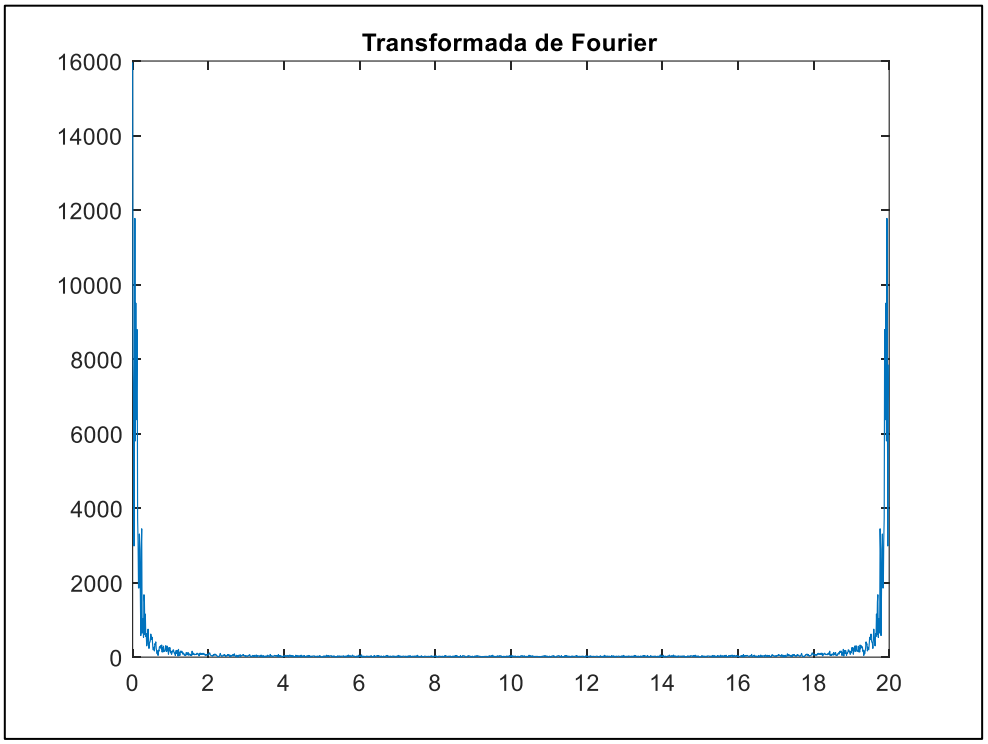


Fig.3.32 Transformada de Fourier

Según se puede observar en la gráfica 1 Hz podría ser la frecuencia de corte ideal para eliminar las pequeñas variaciones de la señal. Por tanto, se pasa a diseñar un filtro de segundo orden con una frecuencia de corte de 1 Hz.

```
Fc=1;%frecuencia corte
Wn=Fc/(Fs/2);
[num,den]=butter(2,Wn,'low');%obtención del filtro paso bajo ("low")
de orden 2
```

El resultado obtenido es el siguiente:

```
num =

    0.0201    0.0402    0.0201


den =

    1.0000   -1.5610    0.6414
```

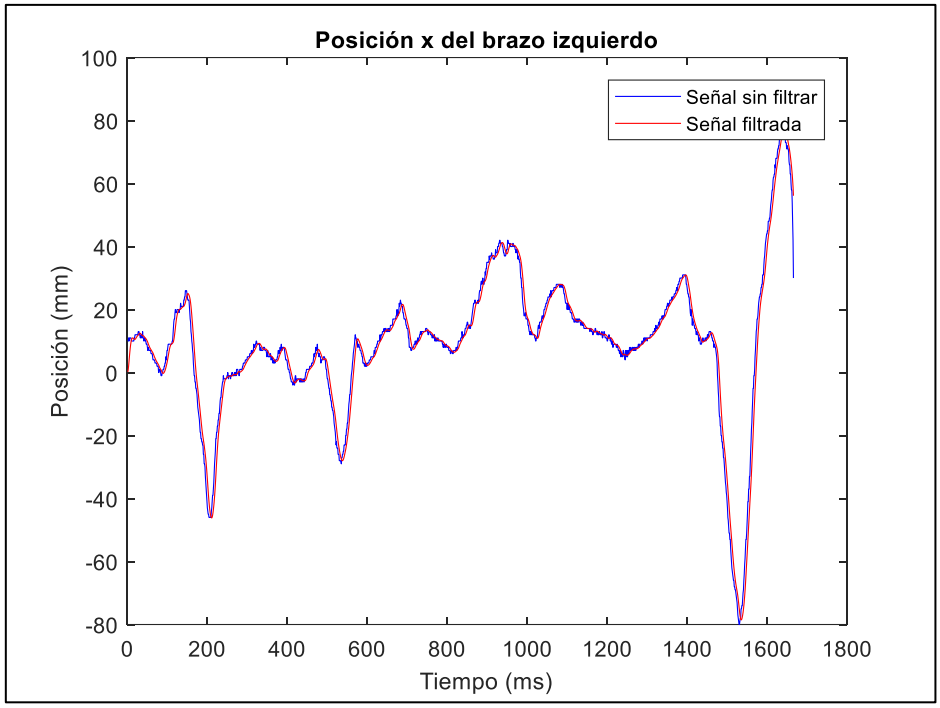


Fig.3.33 Posición con y sin filtrado

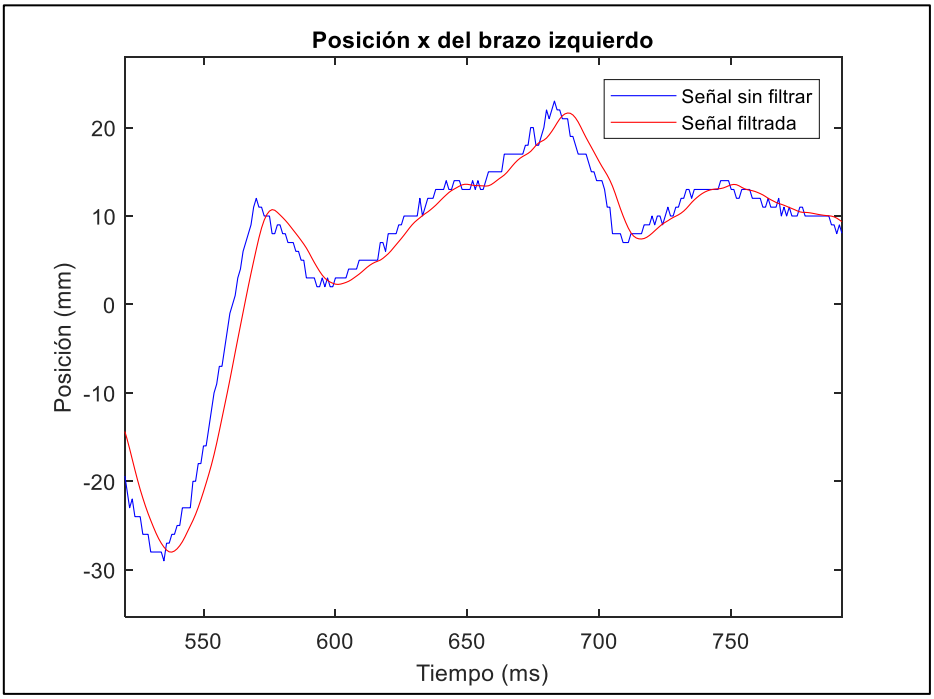


Fig.3.34 Zoom de la posición con y sin filtrado

Este es el primer filtro que se ha implementado en la aplicación. Ha sido capaz de eliminar todo el temblor del robot, pero produce retraso de los movimientos del robot. Este retraso también se puede apreciar en la gráfica anterior donde se puede observar que la señal roja está retrasada respecto de la azul.

Para minimizar este retraso se ha decidido pasar al diseño de un filtro de tercer orden y con una frecuencia de 2 Hz

```
Fc=2;%frecuencia corte

Wn=Fc/(Fs/2);

[num,den]=butter(3,Wn,'low');%obtencion del filtro paso bajo ("low") de orden 3

num =

    0.0181    0.0543    0.0543    0.0181

den =

    1.0000   -1.7600    1.1829   -0.2781
```

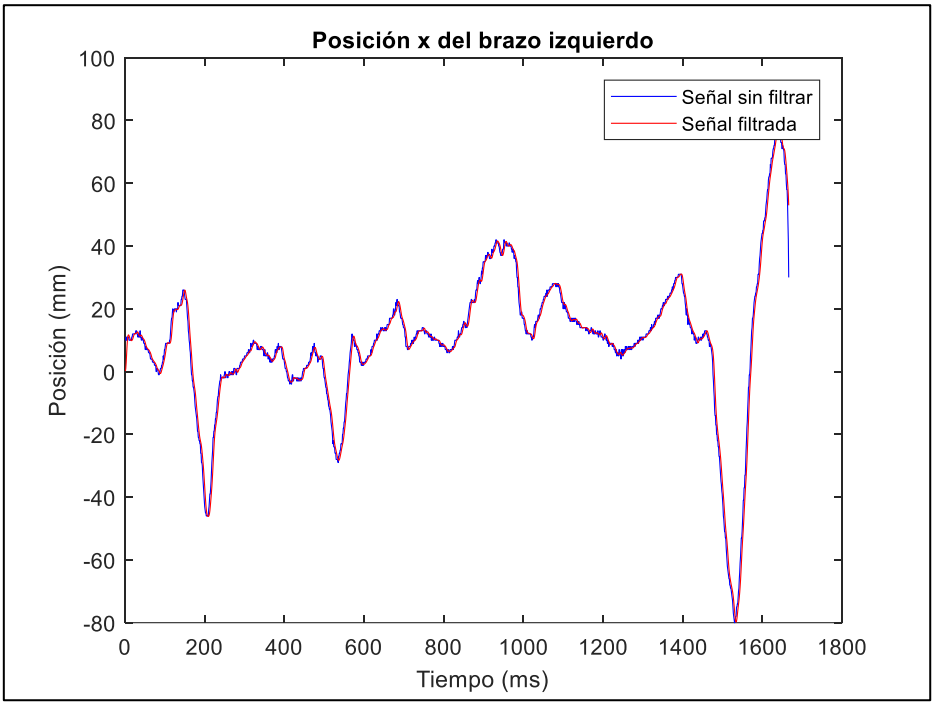


Fig.3.35 Posición con y sin filtrado

Tal y como se puede observar en la gráfica del filtro de tercer orden, este filtro no produce tanto retraso y elimina las pequeñas variaciones de la señal. Este es el filtro que se ha implementado definitivamente en la aplicación. Con este filtro se ha conseguido eliminar el temblor y además el retraso en los movimientos del robot es muy pequeño.

3.7. APLICACIÓN SIN MARCADORES

En uno de los institutos de investigación de la universidad se ha realizado una aplicación que detecta la posición absoluta respecto la cámara de las muñecas de una persona. De esta forma para obtener la posición de las muñecas no hacen falta ponerse marcadores, sino que simplemente mueves los brazos delante de la cámara y el programa detecta la posición.

Comparando esta aplicación con la que detecta marcadores, se puede decir que la ventaja principal es la estética ya que no utiliza marcadores en las muñecas para obtener datos de las posiciones. Pero también presenta desventajas. La principal desventaja es que, una vez iniciado el programa, delante de la cámara solo puede estar la persona que realiza la tele operación con el robot ya que, si entra en la cámara otra persona, la aplicación no sabe cuál es la muñeca a partir de la cual se quiere sacar la información de la posición. Otra de las desventajas sería al tiempo. Se ha podido comprobar que la aplicación de los marcadores tarda 50 ms en el procesamiento para generar las posiciones mientras que con la aplicación sin marcadores se tarde 83 ms. Por tanto, si lo que importa es la rapidez de la aplicación, es mejor utilizar la aplicación con marcadores. En cambio, si lo que interesa es la estética, será mejor utilizar la aplicación sin marcadores.

Esta aplicación envía al robot la posición utilizando las tramas explicadas en el apartado de la aplicación detecta-marcadores. Así pues, se ha programado el robot YuMi y el IRB 140 de igual forma que se ha hecho con la aplicación detecta-marcadores y de esta forma se ha conseguido tele operar el robot sin necesidad de marcadores. La única diferencia respecto a la programación del robot con marcadores, es que la aplicación envía posiciones absolutas respecto la cámara y para mover el robot se necesitan posiciones relativas respecto a un punto inicial y respecto a los ejes del robot.

La solución que se ha encontrado para este problema es que durante 10 iteraciones el robot recibe las posiciones de las muñecas (durante estas iteraciones los brazos de la persona deben estar parados) y se hace una media de estas posiciones. La media será el punto inicial a partir del cual se moverá el robot. Es decir, si respecto al punto inicial (el punto medio calculado) el brazo humano se mueve 10 cm el robot también moverá 10 cm respecto su punto inicial. En este caso la posición inicial del robot es la misma que en la aplicación con los marcadores.

```
PROC calculo_valor_medio()
    j:=0;
    vmedioRX:=0;
    WHILE j<=10 DO
        lectura_socket;
        TPWrite cadena_datos_recibida;
        IF (coorRX<>0) AND (coorRY<>0) AND (coorRZ<>0) AND (
        coorLX<>0) AND (coorLY<>0) AND (coorLZ<>0)THEN


            vmedioRX:=vmedioRX+coorRX;
```

```
            vmedioRY:=vmedioRY+coorRY;
            vmedioRZ:=vmedioRZ+coorRZ;
            vmedioLX:=vmedioLX+coorLX;
            vmedioLY:=vmedioLY+coorLY;
            vmedioLZ:=vmedioLZ+coorLZ;
            coorRX_anterior:=coorRX;
            coorRY_anterior:=coorRY;
            coorRZ_anterior:=coorRZ;
            coorLX_anterior:=coorLX;
            coorLY_anterior:=coorLY;
            coorLZ_anterior:=coorLZ;
            SocketSend socket_client\Str:=cadena_enviar;
            j:=j+1;
        ENDIF
    ENDWHILE
    vinicialRX:=vmedioRX/j;
    vinicialRY:=vmedioRY/j;
    vinicialRZ:=vmedioRZ/j;
    vinicialLX:=vmedioLX/j;
    vinicialLY:=vmedioLY/j;
    vinicialLZ:=vmedioLZ/j;

endproc
```

Otra cosa que se debe tener en cuenta es que los ejes de coordenadas del robot no son los mismos que los de la cámara. La aplicación envía información respecto los ejes de la cámara, por tanto, se necesita pasar la información respecto a los ejes del robot. Las siguientes expresiones son las que se han utilizado para hacer esta transformación, teniendo en cuenta también que se programa para que los movimientos sean en espejo

```
    coorRX:=-coor_cam_RZ;
    coorRY:=-coor_cam_RX;
    coorRZ:=-coor_cam_RY;

    coorLX:=-coor_cam_LZ;
    coorLY:=-coor_cam_LX;
    coorLZ:=-coor_cam_LY;
```

4. CONCLUSIONES Y TRABAJOS FUTUROS

Desarrollando el proyecto se han aprendido cosas nuevas que antes eran desconocidas. En primer lugar, se ha aprendido a usar la visión artificial para obtener la posición de un objeto utilizando marcadores.

En segundo lugar, se reforzaron los conocimientos sobre el lenguaje de programación de RAPID y se aprendió a realizar programación multitarea con puntos de sincronización.

Después de la realización de todos los pasos expuestos anteriormente se ha conseguido alcanzar al objetivo final del proyecto; la tele operación de brazos robóticos.

Como se puede observar a lo largo del proyecto la aplicación desarrollada se puede implementar en varios robots. Esto trae consigo la ventaja de que, si en cualquier momento por algún motivo se quiere cambiar de robot, no hace falta cambiar la programación, sino que se puede pasar a utilizar otro robot sin ningún problema, aprovechando la aplicación desarrollada (siempre que sea de la empresa ABB y utilice el RAPID como lenguaje de programación). Si se quiere utilizar otro robot que no utilice el RAPID basta con seguir el mismo esquema que la programación realizada en RAPID y de una forma fácil se puede utilizar la aplicación en otro robot.

En un futuro utilizando cámaras para observar el movimiento del robot, se podría probar la aplicación estando la persona en una habitación diferente. De esta forma la persona mediante cámaras observará los movimientos del robot y ella moverá sus brazos para que el robot haga esos mismos movimientos con el fin de realizar alguna tarea. Así pues, se tiene la posibilidad de tele operar el robot sin necesidad de estar ni siquiera en el mismo espacio, pudiendo haber distancias elevadas entre el robot y la persona que lo controla.

5. BIBLIOGRAFÍA

5.1. DOCUMENTACIÓN

[1] Aplicaciones de visión artificial. https://blog.infaimon.com/sistemas-de-vision-artificial-tipos-aplicaciones/ (16/03/2019)

[2] Librerías de visión artificial. http://miradadelgolem.blogspot.com/2013/04/librerias-de-vision-artificial.html (16/03/2019)

[3] Realidad aumentada. https://es.wikipedia.org/wiki/Realidad_aumentada (16/03/2019)

[4] ARToolKit. https://es.wikipedia.org/wiki/ARToolKit (18/03/2019)

[5] Tipos de cámaras. https://drescuela.com/tipos-de-camaras/ (20/05/2019)

[6] Cámara 3D. https://es.wikipedia.org/wiki/C%C3%A1mara_estereosc%C3%B3pica (20/05/2019)

[7] Componentes de una cámara. https://es.wikipedia.org/wiki/C%C3%A1mara_fotogr%C3%A1fica (20/05/2019)

[8] Robots colaborativos. https://www.aer-automation.com/mercados-emergentes/robotica-colaborativa/ (12/06/2019)

[9] Robot industrial YuMi. https://new.abb.com/products/robotics/es/robots-industriales/yumi (12/06/2019)

[10] Colegio de ingenieros técnicos industriales de Vizcaya https://www.coitibi.net/serviciosprofesionales/ejercicio-profesional/calcula-precio-hora-de-tu-trabajo (10/06/2018)

[11] Colegio de ingenieros técnicos industriales de Valencia https://copitival.es (10/06/2018)

[12] Claudia Anais. Desarrollo de una aplicación para el guiado automático de un vehículo eléctrico. TFG en Tecnologías Industriales (2016).

[13] Héctor Sánchez. Control de posición y movimiento de sistemas robotizados mediante detección de marcadores con visión artificial. TFG en Tecnologías Industriales (2015).

[14] Emima Jiva. Desarrollo de controladores para la coordinación de robots móviles mediante comunicaciones inalámbricas. TFG Electrónica Industrial y Automática (2018).

[15] Alejandro Juan. Control de robots mediante Raspberry Pi. TFM en Automática e Informática Industrial (2018).

[16] Definición de robot. http://lema.rae.es/dpd/srv/search?key=robot (12/06/2019)

[17] Definición de robótica. https://gl.wikipedia.org/wiki/Rob%C3%B3tica (12/06/2019)

[18] Robot Industrial. https://es.wikipedia.org/wiki/Robot_industrial (12/06/2019)

[19] Robot Industries Association (RIA). Definición de robot industrial. www.robotics.org/ (20/03/2019)

[20] International Organization for Standarization. Definición de robot industrial. www.iso.org

[21] Sistemas Operativos Libres para Robots (2011). http://es.slideshare.net/leoyamasaki/sistemas-operativos-para-robots (16/05/2019)

[22] Everware-CBDI. Definicion de SOA. http://www.cbdiforum.com/ (16/03/2019)

[23] Service Oriented Architecture http://www.msig.espol.edu.ec/recursos/9.Service_Oriented_Architecture_Resumen.pdf

[24] Robot Operating System (ROS). www.ros.org (16/03/2019)

[25] Descarga MinGW https://sourceforge.net/projects/mingw/files/MinGW/Extension/make/mingw32-make-3.80-3/mingw32-make-3.80.0-3.exe/download?use_mirror=netix&download= (16/03/2019)

5.2. IMÁGENES

[1] Robot manipulador IRB 140. http://www.directindustry.es/prod/abb-robotics/product-30265-565894.html (12/06/2019)
[2] Robot móvil de aire. http://www.interempresas.net/Horticola/Articulos/151745-El-uso-de-robots-en-tareas-agricolas.html (12/06/2019)
[3] Robot móvil con ruedas. https://leantec.es/robot-autonomo-esquiva-objetos/ (12/06/2019)
[4] Robot andador. https://www.elpaisdelosjuguetes.es/juguete-educativo-dinosaurios-robot-solar.html (12/06/2019)
[5] RobotStudio. http://webdiis.unizar.es/~raragues/wp/2016/04/ (12/06/2019)
[6] Robot Colaborativo YuMi. https://new.abb.com/products/robotics/industrial-robots/irb-14000-yumi (12/06/2019)
[7] Visión Artificial. http://imechatronic.com/es/vision-artificial/ (12/06/2019)
[8] Torch3Vision. https://pytorch.org/blog/torchvision03/ (12/06/2019)
[9] OpenCV. https://www.pyimagesearch.com/2018/07/19/opencv-tutorial-a-guide-to-learn-opencv/ (12/06/2019)
[10] Realidad Aumentada. https://www.xataka.com/basics/diferencias-entre-realidad-aumentada-realidad-virtual-y-realidad-mixta (12/06/2019)
[11] Escala de la realidad aumentada https://es.wikipedia.org/wiki/Realidad_aumentada
[12] Realidad virtual. https://www.tworeality.com/asi-es-como-la-realidad-virtual-revolucionara-la-educacion-del-futuro/ (12/06/2019)
[13] Realidad aumentada con marcadores. https://aumenta.me/blog/tipos-de-realidad-aumentada/ (12/06/2019)
[14] Cámara compacta. https://www.backmarket.es/camara-foto-panasonic-reacondicionado.html?shopping=gmc&gclid=CjwKCAjwvJvpBRAtEiwAjLuRPacWKdeXCusoT0Cn8BjbO9RWlJhneIjCH773W4FgPHYIOZB4dR7WKxoCPMsQAvD_BwE&gclsrc=aw.ds
[15] Cámara instantánea. https://www.elcorteingles.es/electronica/A22638048-camara-instantanea-fujifilm-instax-mini-9-rosa/ (12/06/2019)
[16] Cámara 3D. https://www.backmarket.es/camara-foto-panasonic-reacondicionado.html?shopping=gmc&gclid=CjwKCAjwvJvpBRAtEiwAjLuRPacWKdeXCusoT0Cn8BjbO9RWlJhneIjCH773W4FgPHYIOZB4dR7WKxoCPMsQAvD_BwE&gclsrc=aw.ds
[17] Visor de una cámara. https://www.xataka.com/fotografia-y-video/aprende-a-comprar-una-camara-de-fotos-ii (12/06/2019)
[18] Diafragma de una cámara. https://www.aulafacil.com/cursos/dibujo-lineal-secundaria/educacion-plastica-y-visual-4-eso/la-camara-reflex-objetivo-diafragma-y-obturador-l15684 (12/06/2019)
[19] Calibrar cámara. https://www.researchgate.net/figure/Conjunto-de-imagenes-utilizado-para-calibrar-la-camara-Todas-ellas-son-imagenes-de-un_fig1_266233234 (12/06/2019)

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Trabajo Fin de Máster en Automática e Informática Industrial

DOCUMENTO 2: PRESUPUESTO

Departamento de Ingeniería de Sistemas y Automática

Universitat Politècnica de València

Valencia, Julio 2019

1. PRESUPUESTO

1.1. NECESIDAD DEL PRESUPUESTO

En gran parte de los proyectos, existe una necesidad de conocer el coste económico que supondría el hecho de poner en marcha el proyecto diseñado ya que de esta forma se puede deducir lo beneficioso que sería reproducirlo. Para sacar conclusiones sobre los beneficios primero se debe sacar el presupuesto de lo que costaría el presupuesto. A continuación, se expondrá el cálculo del presupuesto realizado para este proyecto. Para ello se va a dividir en tres partes: coste del personal, material inventariable y material fungible.

1.2. COSTE DE PERSONAL

Para calcular este apartado se tendrá en cuenta el coste por hora del trabajo de un ingeniero y el número de horas dedicado a la realización del proyecto. Con el fin de obtener el coste del personal se multiplican los dos factores.

El coste por hora del trabajo de un ingeniero se ha considerado de 35€ /h según los datos obtenidos de diferentes colegios técnicos industriales [10],[11].

Dentro de este precio ya se incluyen los costes de la cotización a la Seguridad Social.

TRABAJO	TIEMPO (h)	COSTE (€/h)	TOTAL (€)
Descarga e instalación de la librería de ARUco	2	35	70
Desarrollo aplicación detecta-marcadores	120	35	4.200
Desarrollo del programa en RAPID	80	35	2.800
Puesta en marcha y retoques de la aplicación	20	35	700
Redacción de los documentos	60	35	2.100
TOTAL			**9.870**

1.3. MATERIAL INVENTARIABLE

En esta parte se calcula la amortización de los equipos que deben ser comprados específicamente para el proyecto presupuestado. Según el Centro de apoyo a la Innovación, la Investigación y la Transferencia de Tecnología el periodo de amortización para material informático se considera de 5 años y para el resto de equipos 10 años [12].

Para este cálculo se utiliza la siguiente expresión:

$$\text{Coste} = \frac{\textit{Número de meses de uso}}{\textit{periodo de amortización (meses)}} * \textit{Coste del equipo} * \textit{Porcentaje de uso en el proyecto}$$

Para el cálculo del porcentaje del uso del material se ha calculado teniendo en cuenta el número de horas que se usa, entre el número de horas totales del proyecto.

EQUIPO	TIEMPO USO (meses)	AMORTIZACIÓN (meses)	COSTE (€)	%USO	TOTAL (€)
Ordenador portátil	5	60	1000	100	83,333
Router Wi-Fi	5	120	25	70	0,729
Robot industrial	5	120	40000	70	1.166,677
Cámara 3D	5	120	250	90	9,375
TOTAL					**1.260,081**

1.4. MATERIAL FUNGIBLE

Se trata de aquel material que se ha utilizado en el proyecto y que es de corta vida útil.

MATERIAL	PRECIO (€)
Impresión memoria y marcadores	3
Construcción marcadores	4
Encuadernación	7
Folios	3,5
TOTAL	**17,5**

1.5. RESUMEN DEL PRESUPUESTO

CONCEPTO	TOTAL (€)
Coste de personal	9.870
Material Inventariable	1.260,081
Material Fungible	17,5
TOTAL SIN IVA	11.147,581
IVA (21%)	2.340,992
TOTAL CON IVA	**13.488,57**

Así pues, el presupuesto para la realización de este proyecto es de “TRECE MIL CUATROCIENTOS OCHENTA Y OCHO EUROS CON CINCUENTA Y SIETE CÉNTIMOS “.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Trabajo Fin de Máster en Automática e Informática Industrial

DOCUMENTO 3: ANEXOS

Departamento de Ingeniería de Sistemas y Automática

Universitat Politècnica de València

Valencia, Julio 2019

ANEXO I: MANUAL DE USUARIO

El primer paso a realizar para poder poner en marcha la aplicación es la descarga del MinGW [25].

Una vez finalizada la descarga se eligen las siguientes opciones y se pulsa “Next”:

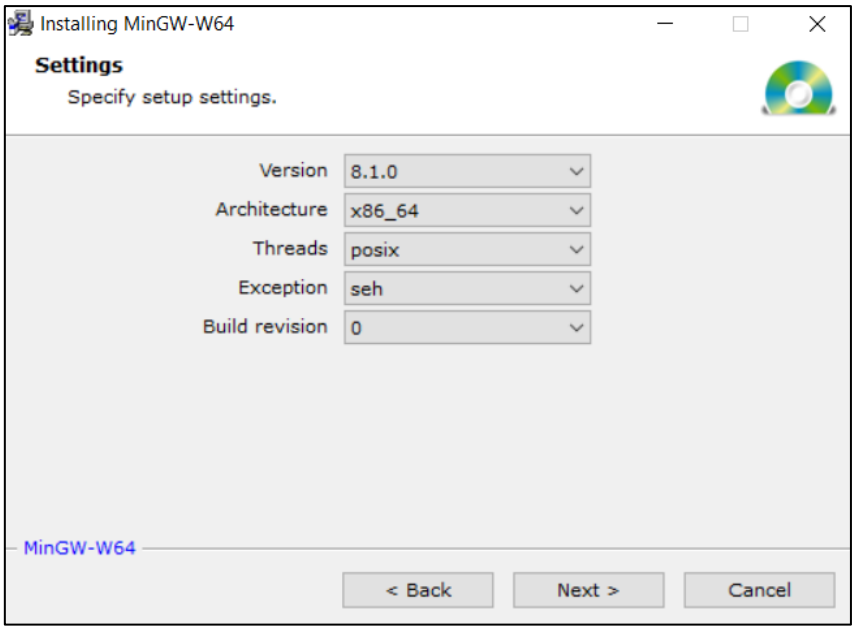


Fig.4 Opciones para la instalación del MinGW

Cuando la instalación se haya terminado, se añade la ruta del MinGW al path del sistema:

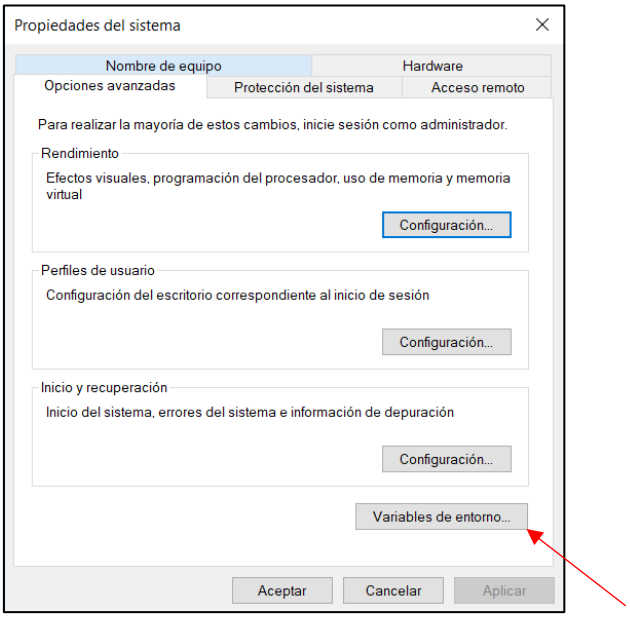


Fig.5 Primer paso para añadir la ruta

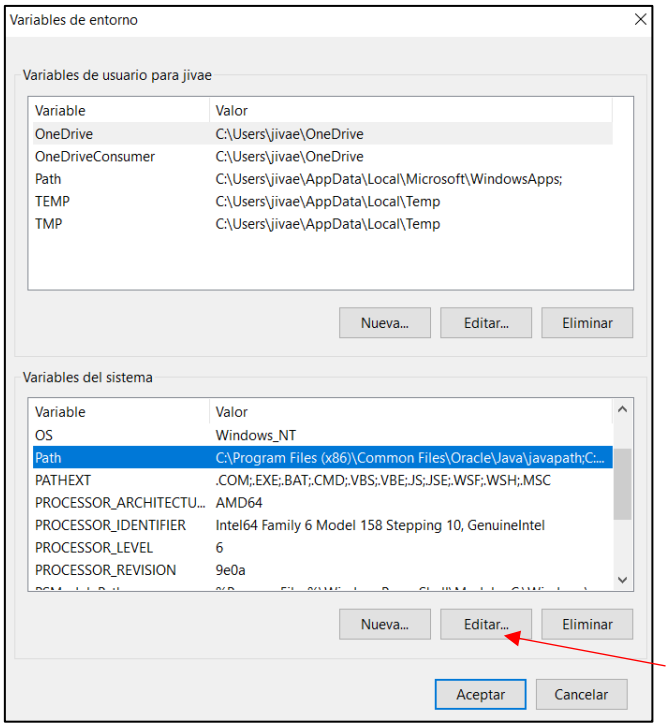


Fig.6 Segundo paso para añadir la ruta

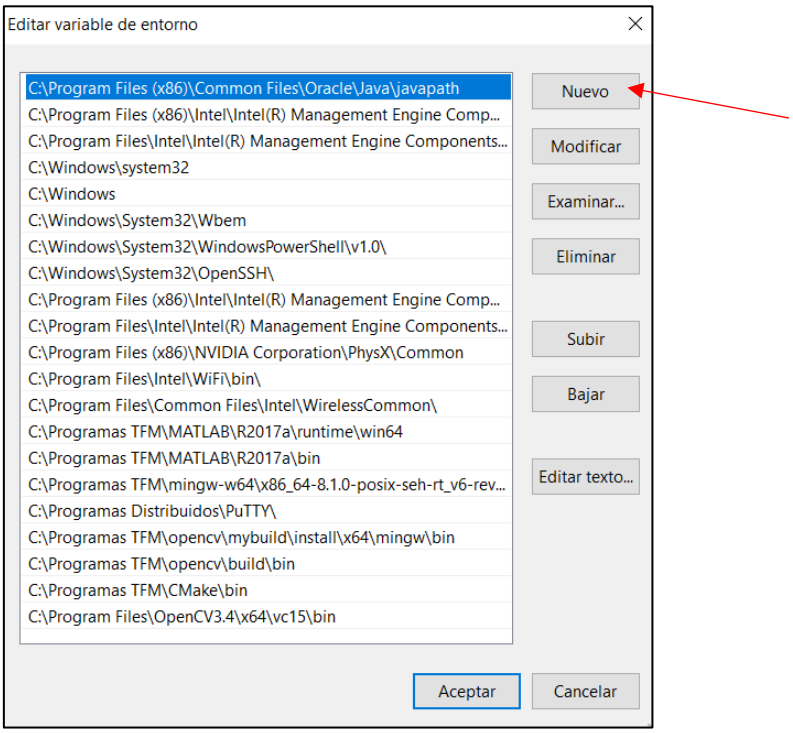


Fig.7 Tercer paso para añadir la ruta

Cuando ya se hayan finalizado estos pasos, se añade la ruta que en este caso es “C:\Programas TFM\mingw-w64\x86_64-8.1.0-posix-seh-rt_v6-rev0\mingw64\bin” y se pulsa “Aceptar” hasta salir de todas las ventanas antes abiertas.

A continuación, se pasa a la calibración de la cámara. Esto se realiza utilizando el mismo ejecutable que para la aplicación detecta-marcadores, pero introduciendo diferentes argumentos. Los pasos de la calibración de la cámara están descritos en el apartado 3.2 de esta misma memoria.

La calibración se hace una sola vez. Cuando se ha terminado de realizar la calibración, en la ventana del programa aparecerá el error que hay en la calibración. Este error debe ser lo menor posible para que de esta forma esté correctamente calibrado.

Una vez terminada la calibración ya se puede poner en marcha la aplicación principal.

El primer paso es poner en marcha el robot con el que se va a trabajar y conectarlo a una red a la que también se tenga acceso con el ordenador que tendrá la aplicación detecta-marcadores.

Se abre el RobotStudio y se añade el controlador disponible en la red.

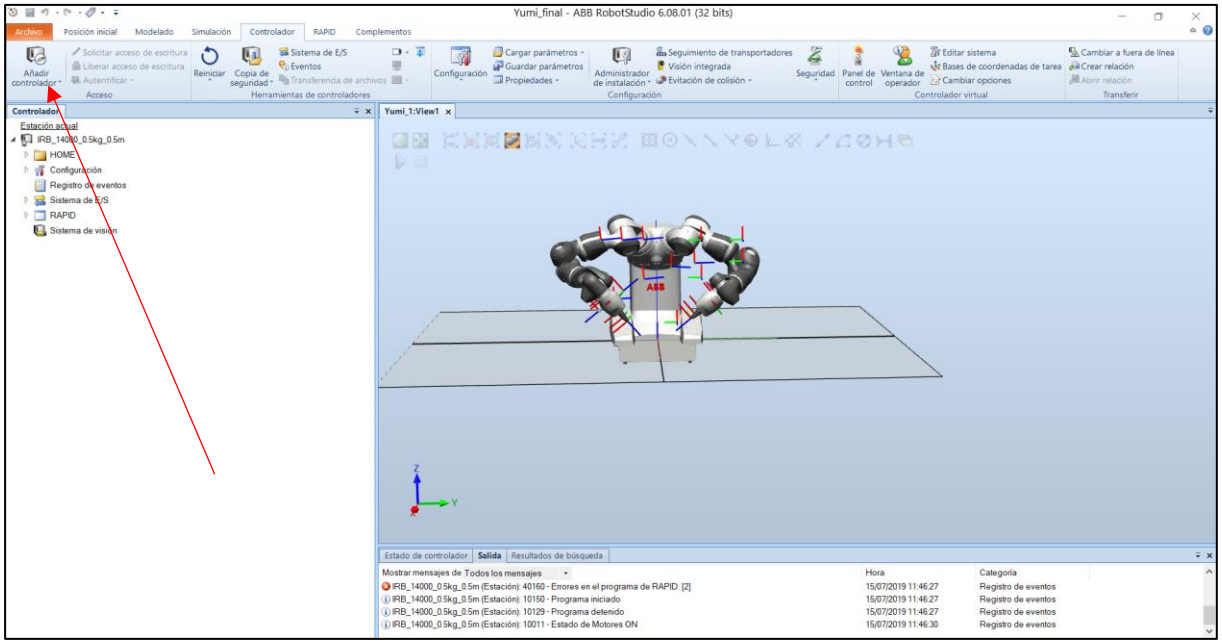


Fig.8 Añadir controlador

Una vez pulsada “Añadir controlador” se elige la opción que añade los controladores disponibles en la red. Allí aparecerá el controlador del robot al que necesitaremos conectarnos. Después de la conexión se cargan los programas (botón derecho sobre la tarea → cargar programa).

Cuando estos pasos ya estén finalizados esto significa que los programas que se han escrito en Rapid ya están dentro del robot.

Cuando se quieran poner en marcha los programas cargados en el robot basta con pulsar el siguiente botón:

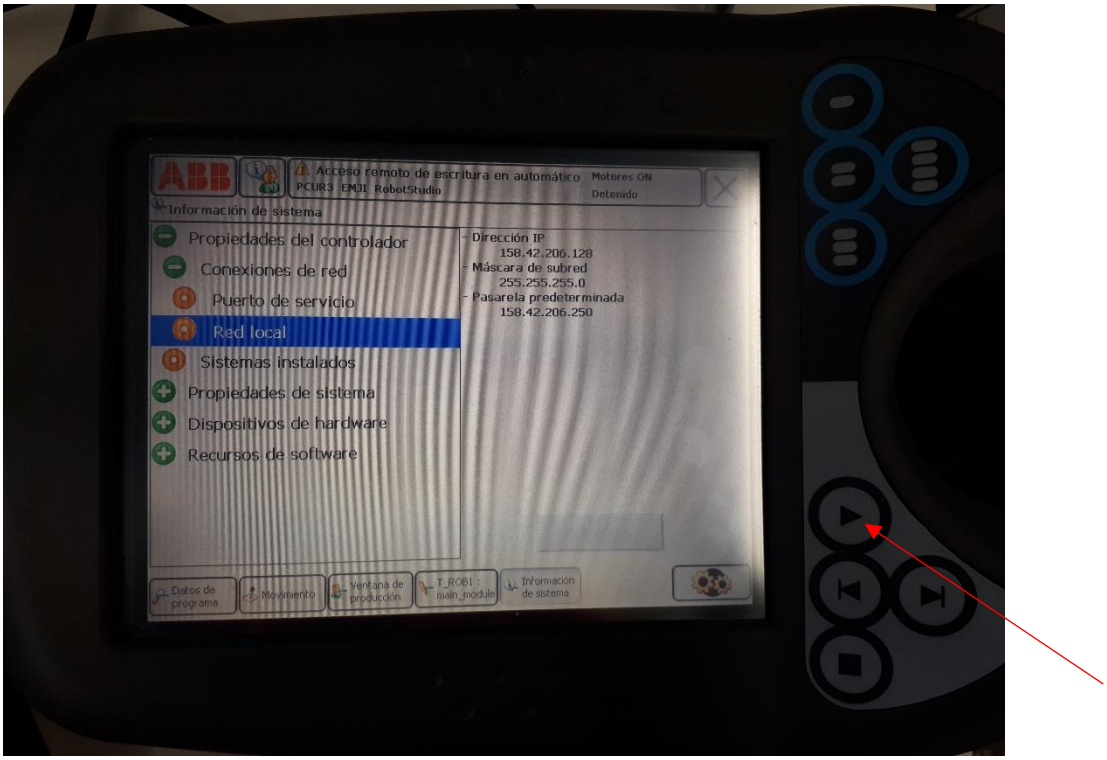


Fig.9 Botón para iniciar programa

Y cuando se quiera finalizar la ejecución del programa, se pulsa el siguiente botón:

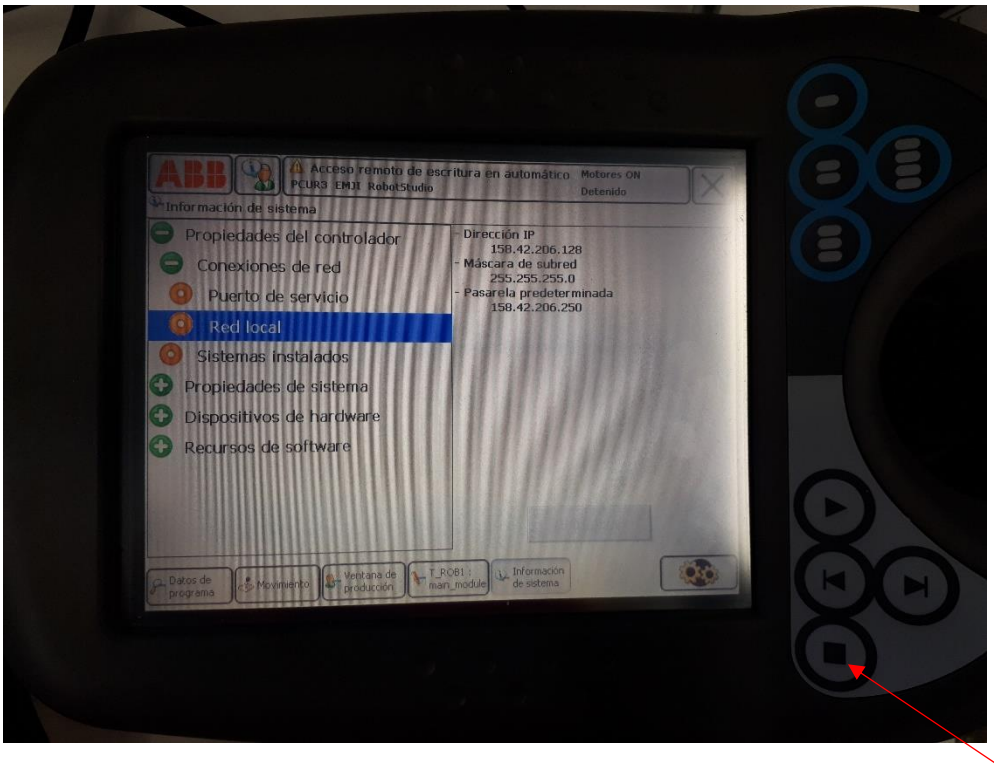
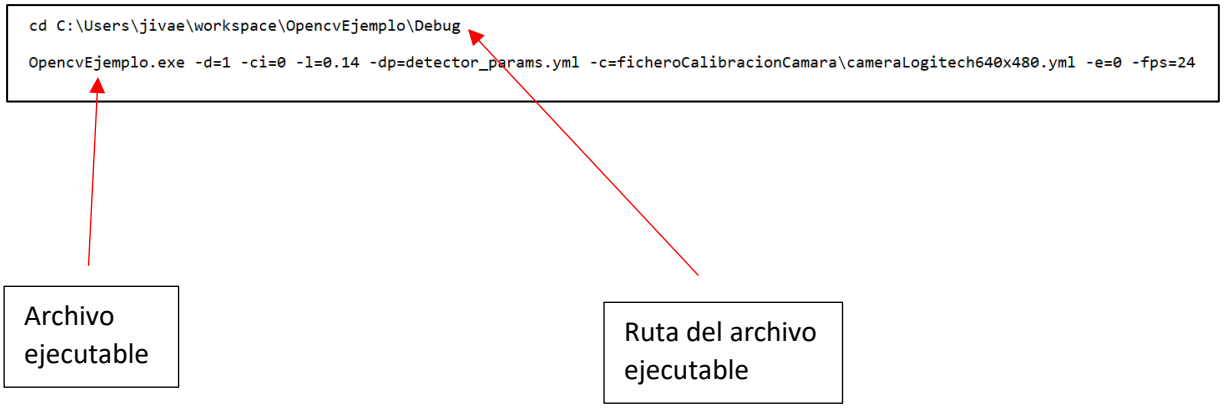


Fig.10 Botón para finalizar programa

Como el robot es el servidor, antes de ejecutar la aplicación detecta-marcadores, se tendrá que poner en marcha para que la aplicación se pueda conectar al robot.

En el caso de la aplicación detecta-marcadores, se tendrá que crear un archivo por lotes para evitar poner los argumentos cada vez que se quiera ejecutar el programa. El archivo por lotes tendrá el siguiente aspecto:



- d→ Diccionario del marcador que se quiere utilizar (en este caso para el IRB 140 se utiliza el diccionario definido como 1 y para el YuMi el diccionario definido com 16).
- ci→ El id de la cámara. En este caso se deshabilitan todas las cámaras y solo se deja la que se quiere utilizar. De esta forma el id de la cámara será 0.
- l →Lo que mide el marcador (en metros).
- dp→Archivo donde se encuentran los parámetros de detección de los marcadores.
- c→Archivo donde se encuentran los parámetros después de la calibración de la cámara.
- e→Si se quiere detectar el marcador se pone 0 y si se quiere calibrar la cámara se pone 2.
- fps→ Frames por segundo.

Para ejecutar el programa, una vez creado el archivo por lotes se pulsa doble click (antes debe estar en marcha el programa del robot).

Una vez ejecutado el archivo se abrirá la siguiente ventana, donde se muestra lo que la cámara detecta:

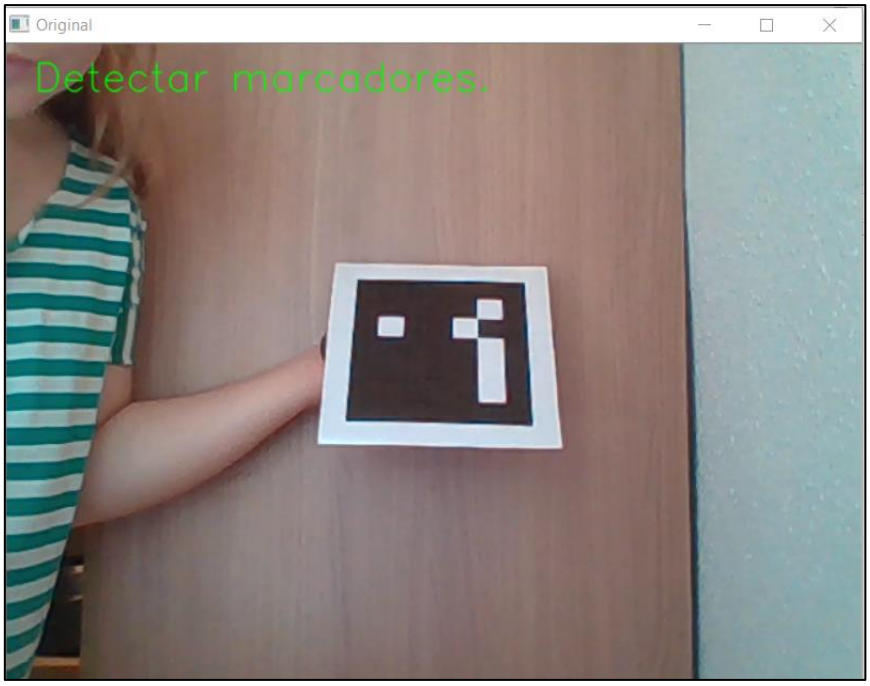


Fig.11 La ventana "Original"

Estando activada la ventana original se pulsa la tecla "v" y aparece otra ventana llamada "Detección" donde se puede observar la detección de los marcadores:

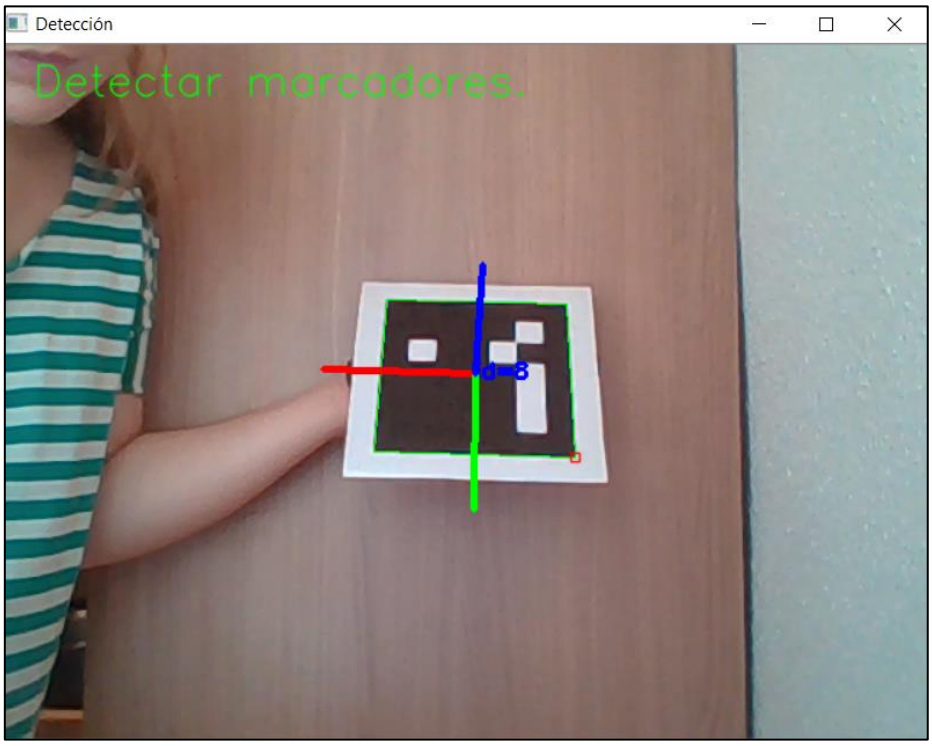


Fig.12 La ventana "Detección"

Estando la ventana “Original” activada y asegurándose que en la ventana “Detección” se observan los marcadores detectados, se pulsa la tecla “s”. A continuación, se esperan 5 segundos para que se guarde la posición inicial respecto de la cual se enviarán las coordenadas al robot. Una vez transcurridos los 5 segundos la persona ya se puede empezar a mover asegurándose siempre que la cámara la pueda detectar. Cuando se empieza a enviar la posición al robot, en la ventana “Original” aparece el texto “Iniciar análisis marcadores”.

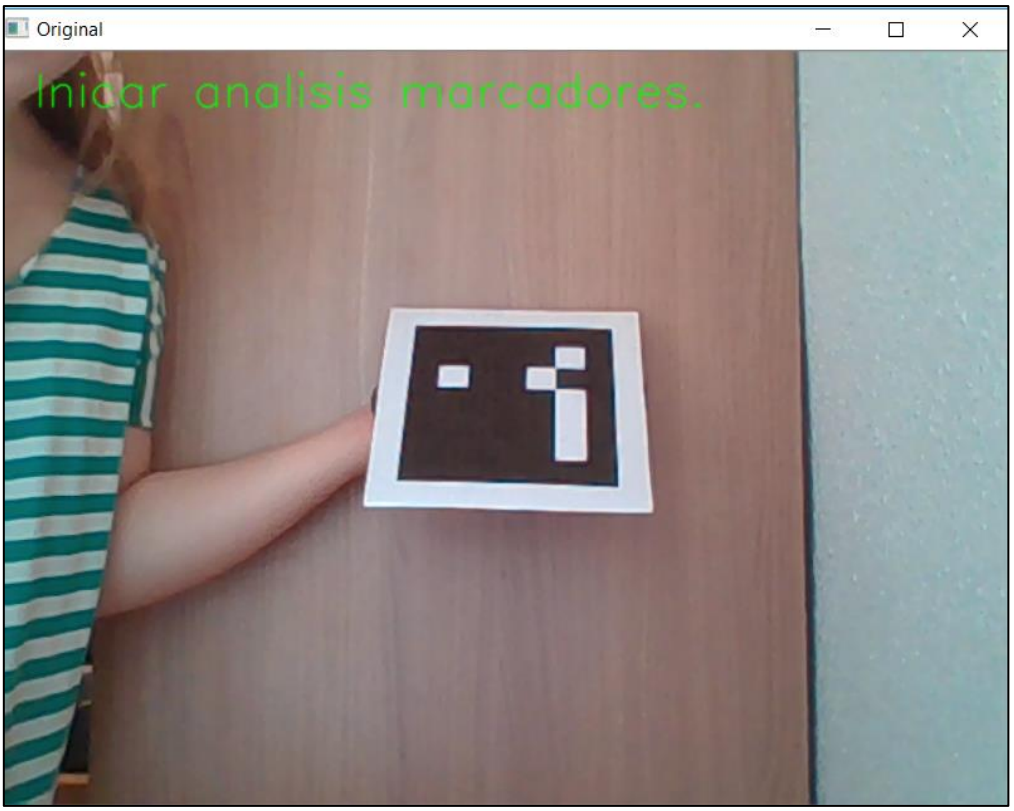


Fig.13 Ventana cuando se envían las posiciones.